

Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

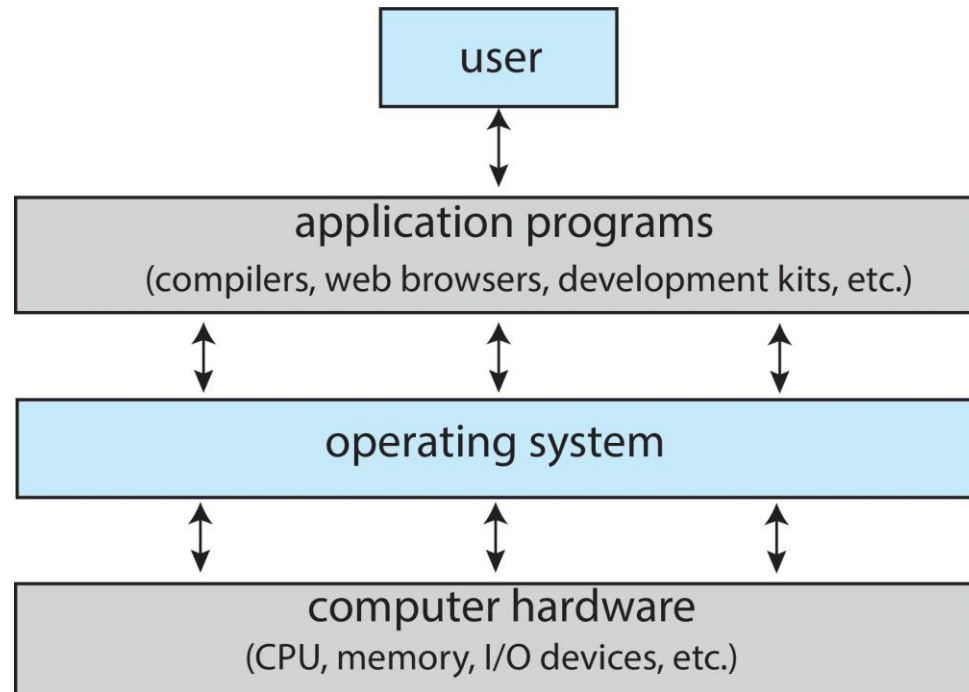
Curs 1 - outline

- structura sistemelor de calcul
- ce este un sistem de operare
- serviciile sistemului de operare
- pornirea sistemului (procesul de boot)
- procesul de login utilizator
- interpretorul de comenzi

Structura sistemelor de calcul

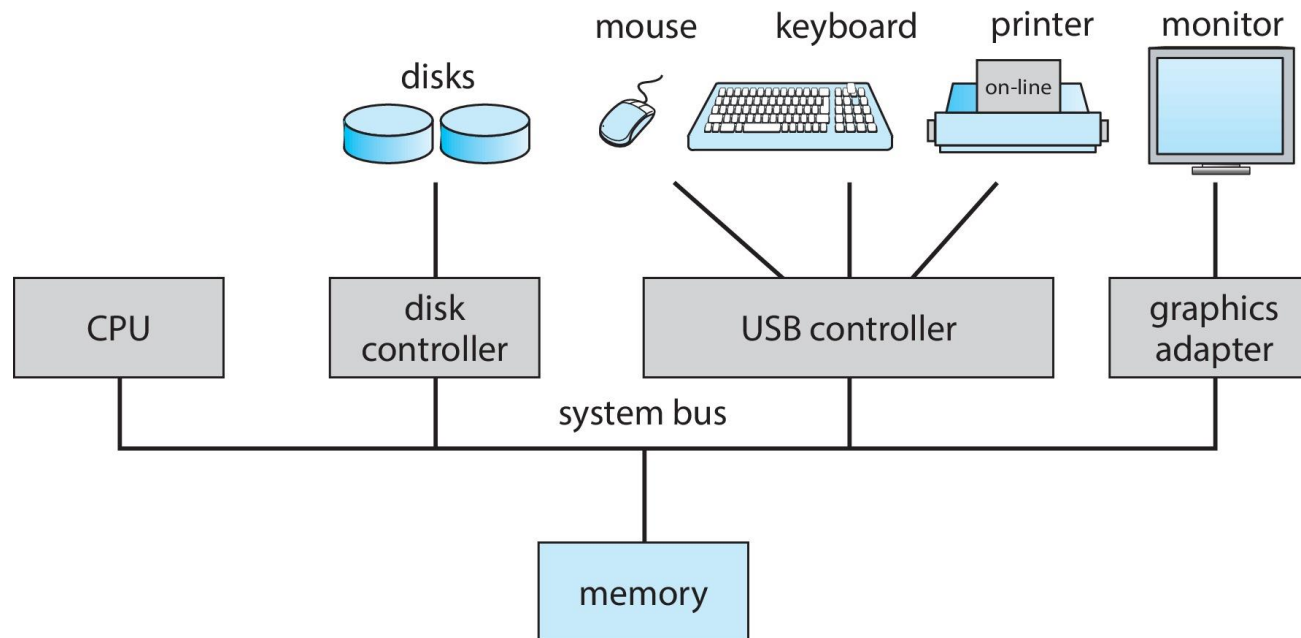
- componente sistem de calcul:
 - Hardware – resursele de calcul de baza
 - CPU, memorie, echipamente intrare/iesire (I/O)
 - Sistem de operare
 - controleaza si coordoneaza utilizarea HW intre programe si utilizatori
 - Programe de aplicatie – definesc modul in care resursele sistemului sunt folosite pentru a rezolva problemele utilizatorilor
 - Procesoare de text, compilatoare, browser-e web, sisteme de baze de date, jocuri video
 - Utilizatori
 - oameni, masini, alte computere

Perspectiva abstracta a componentelor unui calculator

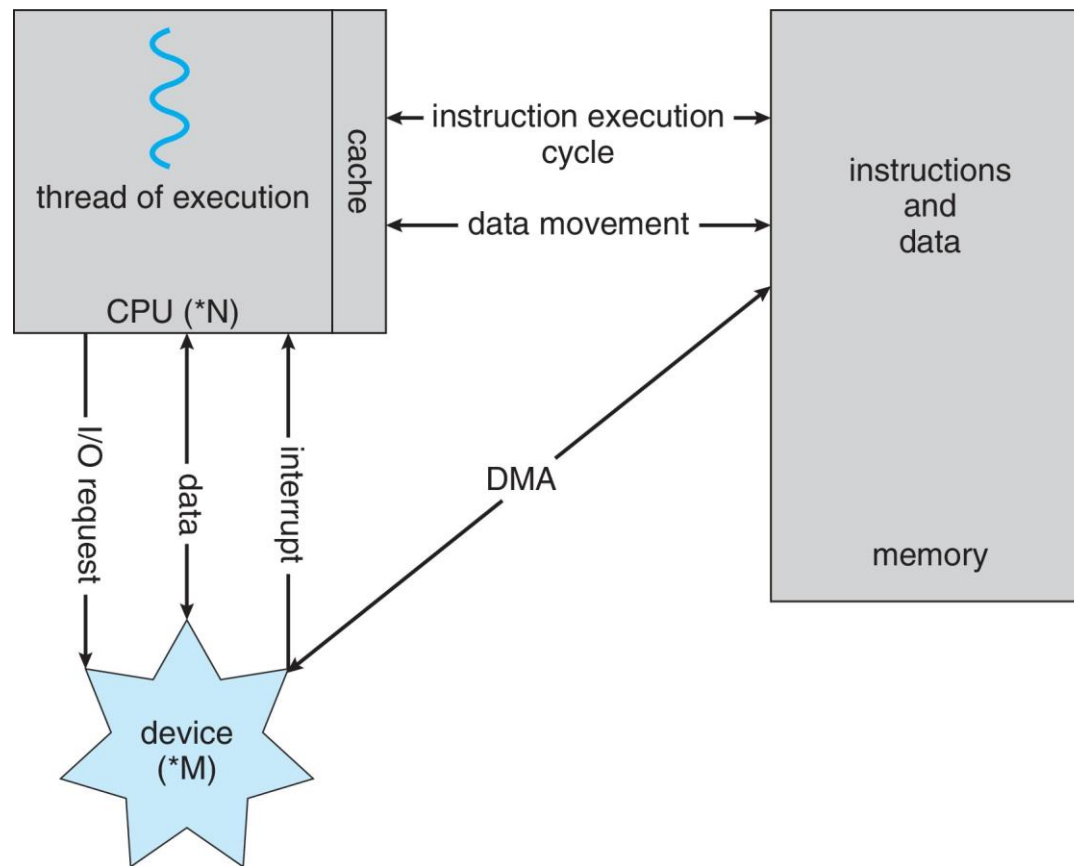


Organizarea sistemelor de calcul

- operarea sistemelor de calcul
 - unul sau mai multe procesoare si controllere de echipamente comunica prin intermediul unei magistrale care asigura accesul la memorie
 - efect net: concurenta executiei procesoarelor si a echipamentelor care intra in competitie pentru cicli de acces la memorie



Cum functioneaza un computer modern



Ce este un sistem de operare?

- un program care intermediaza intre utilizator si HW calculatorului
- obiectivele sistemului de operare:
 - executa programele utilizator si usureaza solutionarea problemelor
 - face sistemul de calcul convenabil de utilizat
 - in particular, responsabil pentru definirea unor abstractii software
 - eg., lucram cu fisiere nu cu blocuri de disc, cu conexiuni de retea nu cu sirurile de biti manipulate de placile de retea
 - foloseste HW computerului in mod eficient

Ce face un sistem de operare

- depinde de punctul de vedere
- utilizatorii vor usurinta utilizarii si performanta
 - nu le pasa de utilizarea resurselor
- dar calculatoarele mari (eventual supercomputere) trebuie sa satisfaca asteptarile tuturor utilizatorilor
 - sistemul de operare = alocator de resurse si program de control care eficientizeaza folosirea HW si gestioneaza executia programelor utilizator
- utilizatorii de statii de lucru au resurse dedicate, dar adesea folosesc resurse partajate de catre servere
- echipamentele mobile (smartphone, tablet) au resurse limitate, sunt optimizate pentru uzabilitate si viata bateriei
 - interfete utilizator speciale, touch screen, recunoastere vocala
- unele computere au interfete limitate sau n-au deloc, eg. embedded systems in echipamente industriale sau automobile
 - in principal ruleaza fara interventia utilizatorului

Definitia sistemului de operare

- nu exista o definitie general acceptata
- “software-ul cu care este echipat calculatorul livrat de producator” e o buna aproximatie
 - variaza insa mult
- “programul care ruleaza in permanenta pe calculator” este nucleul (kernelul) sistemului de operare
- restul este fie
 - program de sistem (livrat cu sistemul de operare, dar nu e parte a nucleului), sau
 - aplicatie, toate programele neasociate cu sistemul de operare
- SO actuale de uz general (GPOS, General Purpose OS) sau pt calcul mobil includ si ***middleware*** – un set de framework-uri software care furnizeaza servicii aditionale dezvoltatorilor de aplicatii cum ar fi baze de date, multimedia, grafica

Serviciile sistemului de operare

- sistemul de operare ofera un mediu de executie pentru programe si servicii pentru programe si utilizatori
- o parte a serviciilor SO furnizeaza functii de asistenta a utilizatorului:
 - **interfata utilizator** – aproape toate SO au interfata utilizatori (UI)
 - variaza: linia de comanda **Command-Line (CLI)**, interfata grafica **Graphics User Interface (GUI)**, **touch-screen**, **Batch**
 - **executia programelor** – SO trebuie sa fie capabil sa incarce un program in memorie si sa-l execute, sa termine executia lui fie normal, fie anormal cu indicarea erorii
 - **operatii de intrare/iesire (I/O)** - un program in executie poate cere I/O, ceea ce poate implica acces la fisiere sau la un echipament I/O
 - **manipularea fisierelor** - sistemul de fisiere este in mod particular interesant pt utilizator (programele au nevoie sa citeasca/scrie fisiere si directoare, sa le creeze si sa le stearga, sa le caute, sa afiseze informatii despre ele, sa gestioneze permisiunile de acces la ele)

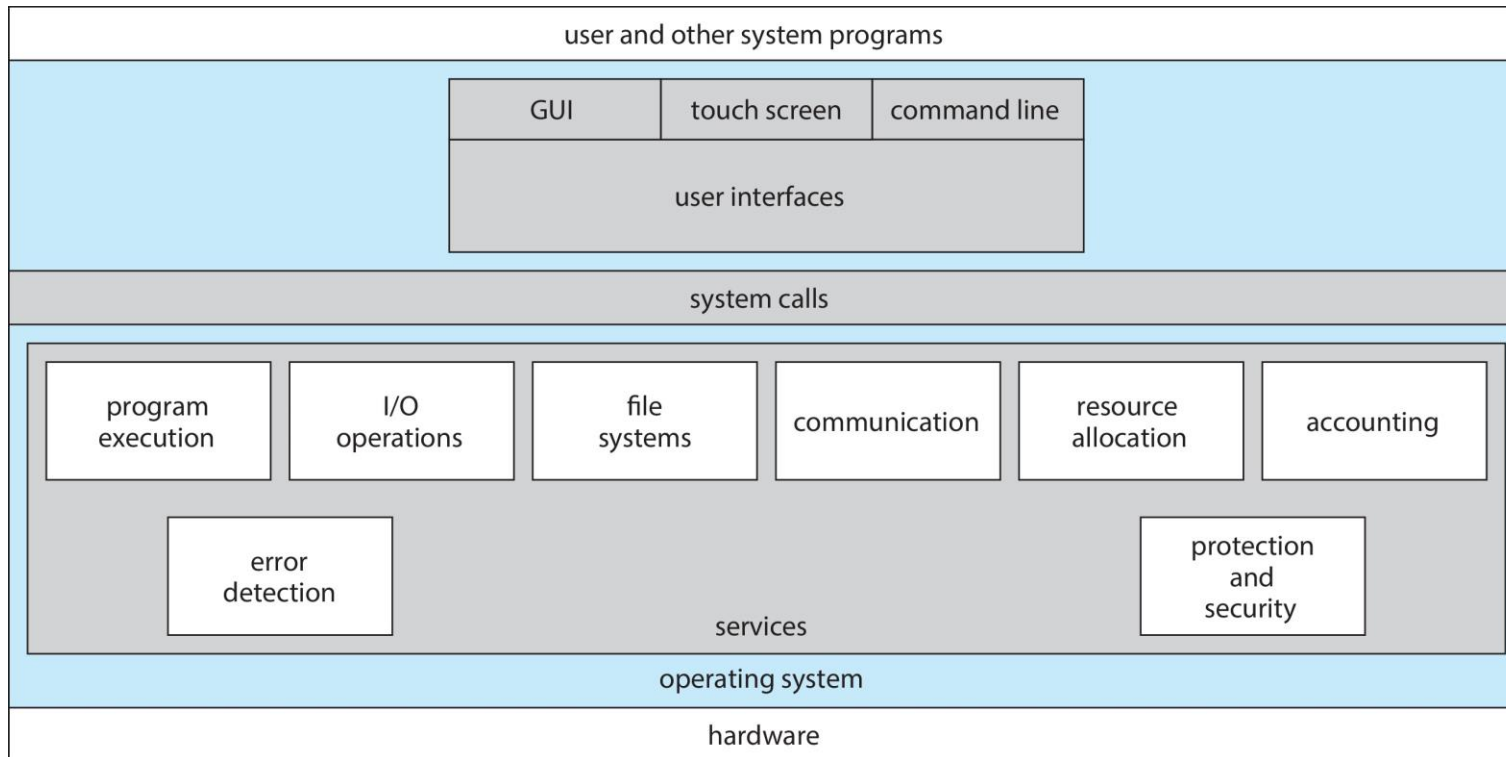
Serviciile SO (cont.)

- o parte a serviciilor SO furnizeaza functii de asistenta a utilizatorului (cont.):
 - **comunicatia** – procesele pot schimba informatii, pe acelasi calculator sau intre calculatoare legate in retea
 - comunicatia poate avea loc prin memorie partajata sau schimb de mesaje (message passing)
 - **detectia erorilor** – SO trebuie sa fie constant constient de posibile erori
 - pot aparea in CPU sau memorie, in echipamente I/O, in programele utilizator
 - pt fiecare tip de eroare, SO trebuie sa ia actiunea potrivita pt a asigura calculul corect si consistent
 - facilitatile de debug pot imbunatati substantial abilitatile utilizatorilor si programatorilor de a utiliza eficient sistemul de calcul

Serviciile SO (cont.)

- alta parte a SO exista pt a asigura operarea eficienta a sistemului in prezenta resurselor partajate
 - **alocarea resurselor** – cand mai multi utilizatori sau programe se executa concurent, au nevoie de resurse fiecare
 - tipuri de resurse - ciclul CPU, memoria principala, stocarea fisierelor, echipamente I/O
 - **logarea executiei** – necesara pt a contabiliza utilizarea resurselor de catre utilizatori si tipul de resurse folosite
 - **protectie si securitate** – informatiile stocate in sisteme multi-utilizator sau conectate in retea pot avea regim de acces restrictionat + procesele concurente nu trebuie sa interfereze unele cu altele
 - **protectia** implica asigurarea ca toate accesele la resursele sistemului sunt controlate
 - **securitatea** sistemului fata de utilizatori externi necesita autentificarea utilizatorilor si se extinde la protejarea echipamentelor I/O externe de incercari de acces invalide

O perspectiva a serviciilor SO



Instalarea si bootarea SO

- SO sunt in general proiectate sa ruleze pe o clasa de sisteme cu o varietate de echipamente periferice
- uzual, SO deja instalat pe calculatorul cumparat
 - se pot insa compila si instala alte SO
 - daca se genereaza un SO de la zero
 - se scrie codul SO
 - se configureaza pt sistemul de calcul pe care va rula
 - se compileaza SO
 - se instaleaza SO
 - se booteaza calculatorul sub comanda noului SO

Exemplu, Linux

- se descarca codul sursa Linux (<http://www.kernel.org>)
- se configureaza nucleul via “make menuconfig”
- se compileaza nucleul folosind “make”
 - se produce vmlinuz, imaginea nucleului
 - se compileaza modulele kernel via “make modules”
 - se instaleaza modulele kernel in vmlinuz via “make modules_install”
 - se instaleaza noul kernel in sistem via “make install”

Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

Outline

- pornirea sistemului (procesul de boot)
- procesul de login utilizator
- interfata cu utilizatorul
- fisiere si directoare

Bootarea sistemului

- la pornirea calculatorului, executia incepe intr-un loc fix din memorie
- SO trebuie sa fie facut disponibil HW ca sa-l poata porni
 - o mica bucata de cod – **bootstrap loader**, **BIOS**, stocat in **ROM** sau **EEPROM** localizeaza kernelul, il incarca in memorie si il porneste
 - uneori e un proces in doi pasi, utilizand un bloc de boot aflat la o adresa fixa in codul ROM, care apoi incarca bootstrap loader-ul de pe disc
 - sistemele moderne inlocuiesc BIOS cu **Unified Extensible Firmware Interface (UEFI)**
- un bootstrap loader uzual este **GRUB**, permite selectia kernelului de pe discuri multiple, cu versiuni si optiuni diferite
- programul kernel se incarca si apoi sistemul ruleaza
- boot loader-erele permit frecvent diferite stari de boot, cum ar fi de pilda single user mode

Procesul de boot Unix

- primul sector al discului de boot (MBR, respectiv succesorul sau GPT)
 - tabela de partitii de disc
 - cod de bootstrap (boot loader)

```
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/4096B
Partition Table: msdos
Disk Flags:
```

Number	Start	End	Size	Type	File system	Flags
1	1049kB	1574MB	1573MB	primary	ntfs	boot
2	1574MB	211GB	209GB	primary	ntfs	
4	211GB	481GB	270GB	extended		
6	211GB	465GB	254GB	logical	ext4	
5	465GB	481GB	16.4GB	logical	linux-swap(v1)	
3	481GB	500GB	18.9GB	primary	ntfs	

Procesul de boot Unix (cont.)

- loader-ul identifica partitia de boot si incarca codul kernel (nucleul sistemului de operare)
- obs: la acest nivel nu exista notiunea (abstractia) de fisier, ci doar sectoare de disc => 2 solutii posibile
 - 1. loader-ul cunoaste o harta a sectoarelor de disc care contin codul kernel
 - solutie hardcoded, implica actualizari ale hartilor atunci cand imaginea kernelului pe disc se schimba, la defragmentarea discului, etc
 - 2. loader-ul are acces la drivere care inteleg structura sistemului de fisiere de pe disc si pot identifica astfel kernelul ca pe un fisier oarecare (folosind calea fisierului)
- ex boot loaders Linux: Lilo, Grub

Procesul de boot (cont.)

- fisierul cu imaginea kernelului (eg, */boot/vmlinuz* pt Linux) se incarca in memorie si kernelul preia controlul masinii HW
- subsecvent, kernelul executa:
 - secventa de intializare a componentelor HW
 - instantiaza principalele componente: controlul proceselor, gestiunea memoriei, gestiunea fisierelor, accounting, gestiunea timpului sistem, mecanismele de protectie HW si de securitate, etc
 - ramane rezident in memorie in asteptarea unor evenimente externe (“program interrupt-driven”)
 - la sfarsitul secventei de initializare executa primul proces (ID = 1): */sbin/init*
 - *init* seteaza modul de operare (*runlevel*)
 - defineste starea masinii de calcul dupa boot
 - traditional definit de un numar intre 0 si 6
 - istoric (sistemele Unix), *init* cauta runlevel-ul in */etc/inittab*
 - apoi, apeleaza scripturi de initializare a serviciilor sistem cf. runlevelului selectat
/etc/rc0.d/, /etc/rc1.d, ..., /etc/rc6.d, /etc/rcS.d

Runlevels

- asignate modului de operare al masinii
 - 0 , power-off
 - 1, single-user mode
 - 2, multi-user fara retea
 - 3, multi-user cu retea dar fara interfata grafica
 - 4, in general nedefinit, rezervat pt utilizari speciale
 - 5, multi-user cu retea si interfata grafica
 - 6, reboot
- Linux: *init* s.n. *systemd*, iar runlevel-urile sunt definite ca *targets*, manipulate cu comenzi specifice (*systemctl*)
- comenzi care manipuleaza runlevels:

\$ runlevel	# afiseaza runlevelul current, similar cu “who -r”
\$ telinit <runlevel>	# comuta kernelul in runlevelul specificat
\$ telinit 6	# reboot

Sisteme cu sau fara GUI

- *init* este responsabil si pt. pornirea proceselor de login pt utilizator:
 - in functie de runlevel: */sbin/getty* respectiv desktop manager-ul de interfete grafice de tip X Window (*xdm*, *gdm*, etc)
 - runlevel 3: *init* porneste *getty* pe un numar prestabilit de terminale
 - runlevel 5: *init* porneste *getty* + desktop manager
 - istoric (sistemele Unix), *init* cauta in */etc/inittab* asocierea terminal – program de login (*getty* sau *xdm/gdm*)
- comutarea sistemului intre runleveluri cu sau fara interfata grafica

\$ telinit 3 # dezactiveaza GUI

\$ telinit 5 # activeaza GUI inapoi

Obs: combinatii de taste (gen Ctrl-Alt-F1 in Linux) permit comutarea intre terminale si GUI (uzual, Ctrl-Alt-F7) in runlevel 5, dar nu se dezactiveaza GUI !

Logarea utilizatorului in sisteme fara GUI

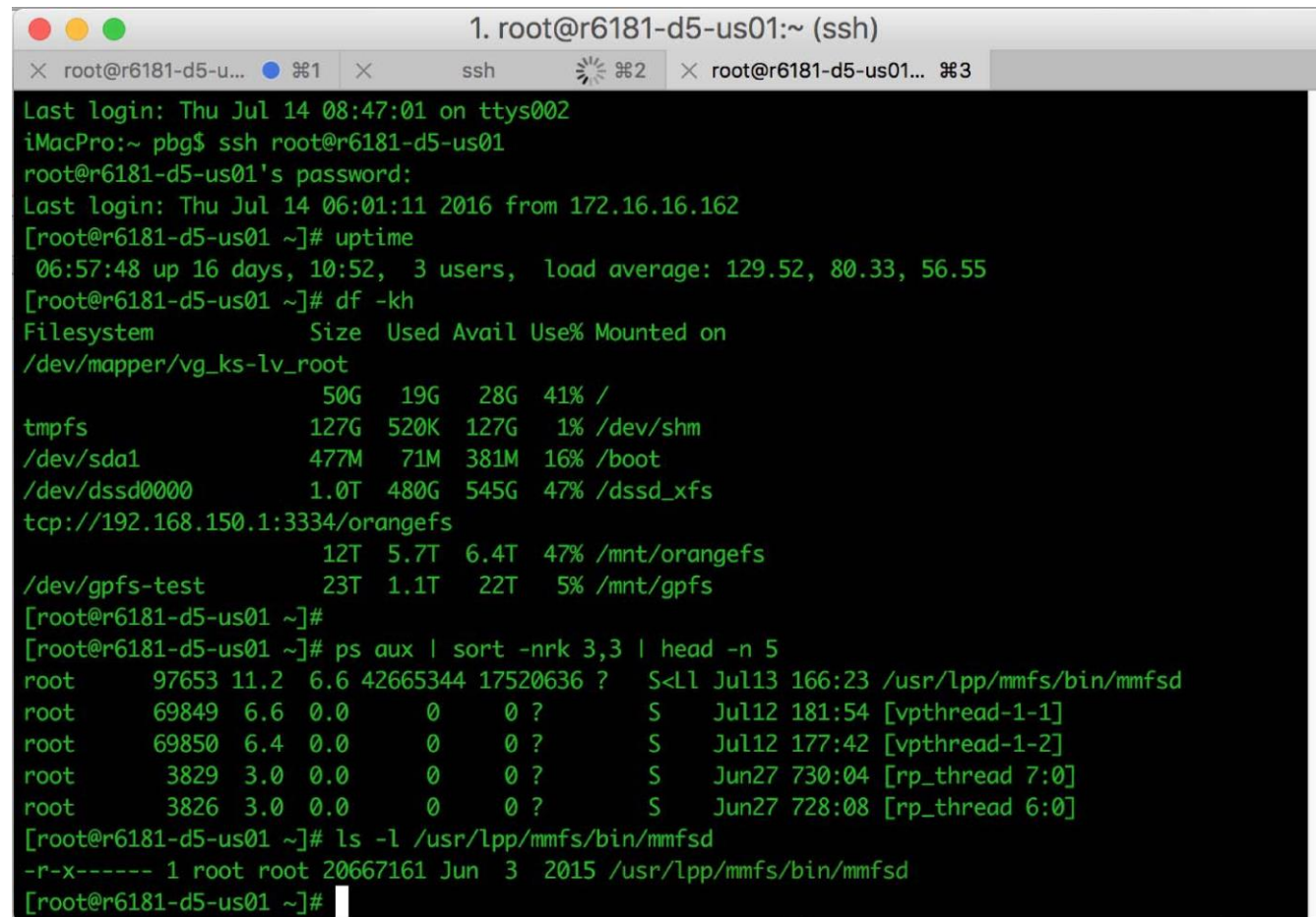
- *getty* afiseaza prompt-ul de login
- utilizatorul introduce numele de utilizator
- *getty* apeleaza */bin/login* care stabileste o noua sesiune de lucru
- *login* afiseaza promptul de parola
 - cauta in */etc/passwd* o intrare corespunzatoare numelui de utilizator
 - verifica parola (de regula stocata criptat in alt fisier, eg. */etc/shadow*)
 - pt parola corecta, executa interpretorul de comenzi (*shell*-ul) asociat intrarii identificate
 - asociaza cu *shell*-ul variabile de mediu (environment)
 - unele variabile importante (USER, SHELL, HOME) initializate cu valorile din campurile citite din intrarea corespunzatoare din */etc/passwd*
- *shell*-ul afiseaza un prompt specific (eg, \$) si asteapta comenzile utilizatorului
- *init* monitorizeaza sesiunea de lucru a utilizatorului
 - la terminarea activitatii (*shell* exit), reporneste o instanta a programului *getty* pe terminalul respectiv

Interpretorul de comenzi

- Command Line Interpreter (CLI), permite introducerea directa a comenzilor
 - program de sistem care preia comenzile utilizator si le executa
 - utilizabil deopotriva in mod interactiv cat si batch (folosind *shell script*-uri)
 - executa atat comenzi interne (executate in cadrul interpretorului) cat si externe (programe incarcate de pe disc)
 - functionalitati principale
 - asigurarea unui mediu de lucru utilizatorului (v. comanda *env*)
 - comenzi de manipulare a fisierelor si directoarelor
 - comenzi de control al executiei programelor
 - controlul si monitorizarea activitatilor de I/O
 - administrarea sistemului (rezervata unui utilizator special cunoscut in mod uzual sub numele de *root*, cu `UID = 0`, v. prima intrare din */etc/passwd*)
 - *samd.*
- ex. interpretoare de comenzi: Bourne Shell (`/bin/sh`), Bourne Again Shell (`/bin/bash`), C Shell (`/bin/csh`), Korn Shell (`/bin/ksh`), etc.

`/etc/shells` contine interpretoarele de comenzi disponibile pe sistem

Bourne Shell (CLI)

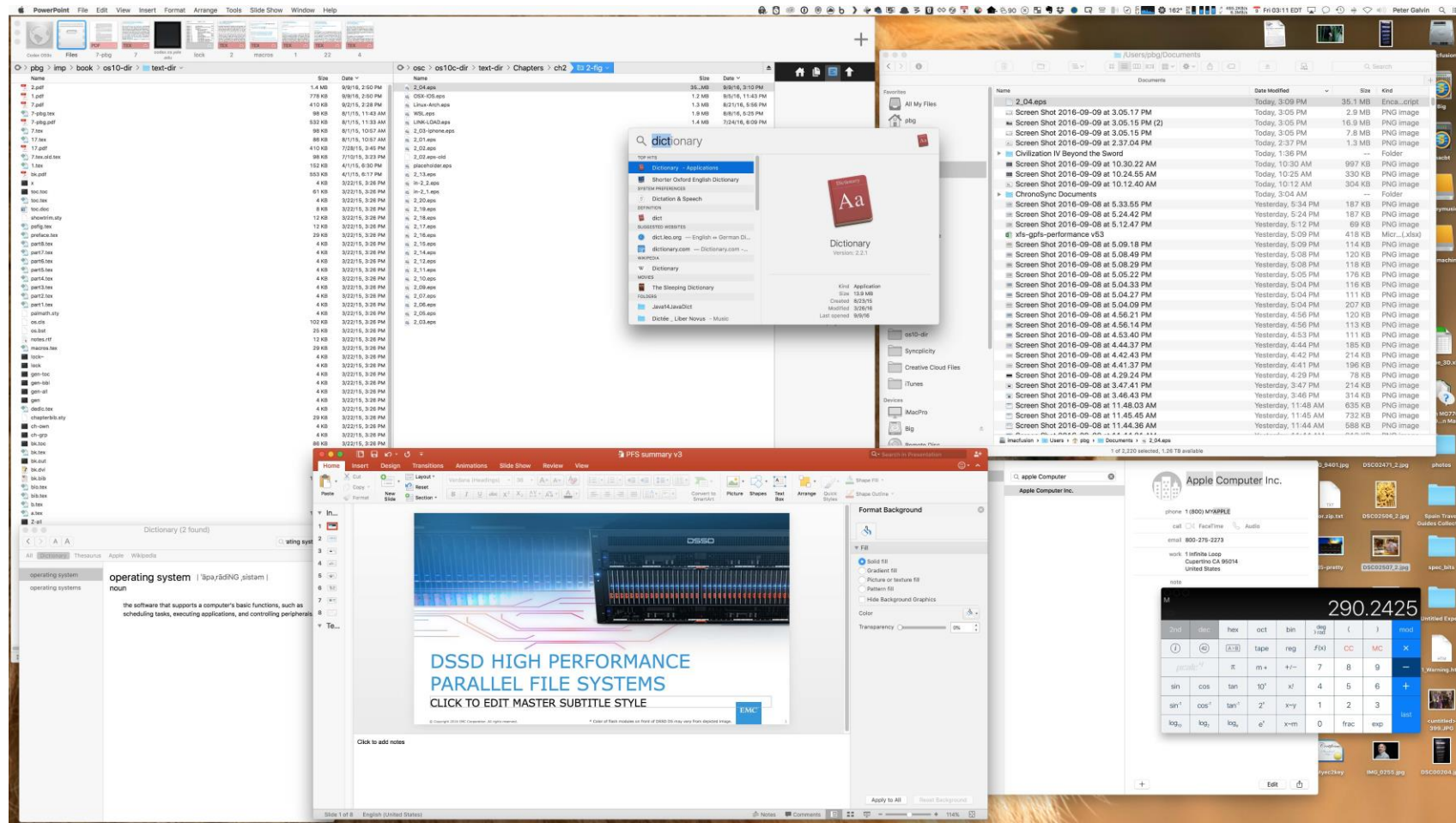


```
1. root@r6181-d5-us01:~ (ssh)
root@r6181-d5-us01:~ (ssh)
Last login: Thu Jul 14 08:47:01 on ttys002
iMacPro:~ pbg$ ssh root@r6181-d5-us01
root@r6181-d5-us01's password:
Last login: Thu Jul 14 06:01:11 2016 from 172.16.16.162
[root@r6181-d5-us01 ~]# uptime
 06:57:48 up 16 days, 10:52,  3 users,  load average: 129.52, 80.33, 56.55
[root@r6181-d5-us01 ~]# df -kh
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vg_ks-lv_root
 50G   19G   28G  41% /
tmpfs           127G  520K  127G   1% /dev/shm
/dev/sda1       477M   71M  381M  16% /boot
/dev/dssd0000   1.0T  480G  545G  47% /dssd_xfs
tcp://192.168.150.1:3334/orangefs
 12T   5.7T   6.4T  47% /mnt/orangefs
/dev/gpfs-test  23T   1.1T   22T   5% /mnt/gpfs
[root@r6181-d5-us01 ~]#
[root@r6181-d5-us01 ~]# ps aux | sort -nrk 3,3 | head -n 5
root      97653 11.2  6.6 42665344 17520636 ?    S<Ll  Jul13 166:23 /usr/lpp/mmfs/bin/mmfsd
root      69849  6.6  0.0      0      0 ?        S    Jul12 181:54 [vpthread-1-1]
root      69850  6.4  0.0      0      0 ?        S    Jul12 177:42 [vpthread-1-2]
root       3829  3.0  0.0      0      0 ?        S    Jun27 730:04 [rp_thread 7:0]
root       3826  3.0  0.0      0      0 ?        S    Jun27 728:08 [rp_thread 6:0]
[root@r6181-d5-us01 ~]# ls -l /usr/lpp/mmfs/bin/mmfsd
-r-x----- 1 root root 20667161 Jun  3  2015 /usr/lpp/mmfs/bin/mmfsd
[root@r6181-d5-us01 ~]#
```

Interfata grafica- GUI

- interfata user-friendly
 - compusa uzual din mouse, tastatura, si monitor
 - **icoanele** reprezinta fisierele, programele, actiuni, etc
 - actionarea butoanelor mouse peste obiecte din interfata determina diverse actiuni (furnizare de informatii, optiuni, executia de functii, deschiderea de directoare, etc)
 - inventata la Xerox PARC
- multe sisteme de azi includ atat CLI cat si GUI
 - Microsoft Windows are GUI si CLI “command” shell
 - Apple Mac OS X are GUI cu kernel UNIX dedesubt si shell-uri disponibile
 - Unix si Linux au CLI (shell-uri) cu GUI optional (CDE, KDE, GNOME)

Mac OS X GUI



Identificarea utilizatorului

- la login, utilizatorul primește un ID propriu (valoare întreaga nenegativă prin care utilizatorul este identificat în SO), user ID-ul (UID)
 - obținut din intrarea corespunzătoare utilizatorului din */etc/passwd*
 - unic
 - asignat de către *root*, singurul care are permisiunea de a scrie în */etc/passwd*
 - nu poate fi schimbat de către utilizator
 - folosit de kernel pentru a verifica dacă procesele utilizatorului au dreptul să execute anumite operații
 - UID = 0 rezervat pt *root* sau *superuser* (administratorul sistemului)
- procesele *root* au privilegii de superuser și de regulă circumventează verificările pe care kernelul le face pentru o serie de operații
- unele dintre funcțiile kernelului pot fi executate doar de procese *root*
- *root*-ul are control total asupra sistemului de calcul
 - Obs: din acest motiv, este puternic descurajată inițiativa utilizatorilor sistemului care știu parola de *root* să ruleze programe obișnuite în calitate de *root* (UID = 0)

Identificarea utilizatorului (cont.)

- la login utilizatorul primește și un GID (Group ID), setat tot de *root* în intrarea corespunzătoare din */etc/passwd*
 - permite partajarea de resurse între membrii aceluiași grup, chiar dacă au UID-uri diferite
 - în schimb, utilizatorii cu GID diferit nu pot accesa aceste resurse partajate ale grupului
 - ex: intrările de director pentru fiecare fișier din sistem conțin perechea (UID,GID) a proprietarului fișierului respectiv
 - comanda shell *ls -l* permite afișarea ID-urilor proprietarului fișierului
- */etc/group*
 - asignează nume lizibile GID-urilor utilizator
 - modificabil doar de către *root*
 - listează și *supplementary GIDs*, i.e. același utilizator poate avea mai multe GID-uri (poate aparține mai multor grupuri)
- */usr/bin/id* afișează UID/GID

\$ id	# UID/GID pt utilizatorul shell-ului
\$ id root	# UID/GID pt alt utilizator (root)

Fisiere si directoare

- **fisier**: abstractie de nivel SO pentru stocarea permanenta a datelor
 - ascunde detaliile stocarii efective a datelor pe disc
 - model usor de inteles al structurii datelor (eg, stream de octeti in Unix)
 - grupate in directoare
 - referite prin nume (poate contine orice caracter mai putin '/')
 - attribute: tip, dimensiune, proprietar, permisiuni, timpul ultimei modificari, etc
 - eg Unix, comanda uzuala pentru afisarea atributelor

```
$ ls -l <nume-fisier>
```

- **director/folder**: colectie de fisiere
 - poate contine alte directoare (subdirectoare)
 - modalitate de a organiza informatia, uzual de-o maniera ierarhica

Fisiere si directoare (cont.)

- sistemele tip Unix folosesc o structura ierarhica de directoare
 - incepe dintr-un director special numit *root* (radacina), desemnat prin caracterul “/”
- directoare speciale create automat atunci cand se creeaza un nou director
 - . directorul curent (directorul nou creat)
 - .. directorul parinte (directorul in care a fost inserata o noua intrare corespunzatoare noului director creat)
 - in cazul directorului radacina (*root*) ./ si ../ reprezinta acelasi director, si anume “/”
- cale (path): secventa de nume de fisiere separate de caracterul /
 - cai absolute: incep intotdeauna cu /
 - cai relative: nu incep cu /, fiind interpretate relativ la directorul de lucru curent (*current working directory*)
- la login, directorul de lucru curent este setat la valoarea obtinuta din */etc/passwd* pentru utilizatorul logat (s.n. *home directory*)
- comanda de tiparire a intrarilor intr-un director: */bin/ls*
 - Obs: un program executabil este si el reprezentat printr-o cale in sistemul de fisiere

Sistemul de fisiere

- componenta speciala a SO care gestioneaza fisierele si directoarele
- structureaza datele pe disc intr-un anumit *format*
- ofera utilizatorului o interfata uniforma de acces la date
 - eg Unix, ierarhie arborescenta de directoare, cu o radacina comuna
- SO moderne sunt capabile sa integreze sisteme de fisiere cu format diferit in aceeasi ierarhie de directoare
 - VFS – Virtual Filesystem Switch (ext3, ext4, ntfs, vfat, etc)
- devin disponibile utilizatorului ca urmare a operatiei de *mount*

```
$ mount -t ext4 /dev/sda1 /
```

- directorul in care se instaleaza discul formatat s.n. *mountpoint*
- */etc/fstab*: tabela system-wide cu mountpoint-uri inspectata la bootarea SO
 - la bootare, mountpoint-urile din tabela se instaleaza ca si cand s-ar fi apelat

```
$ mount -a
```

Mountpoints

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            7.7G     0  7.7G   0% /dev
tmpfs           1.6G   9.6M   1.6G   1% /run
/dev/sda6       233G  218G   3.5G  99% /
tmpfs           7.7G  221M   7.5G   3% /dev/shm
tmpfs           5.0M   4.0K   5.0M   1% /run/lock
tmpfs           7.7G     0  7.7G   0% /sys/fs/cgroup
tmpfs           1.6G   72K   1.6G   1% /run/user/1000
$ mount -t ntfs /dev/sda2 /mnt
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            7.7G     0  7.7G   0% /dev
tmpfs           1.6G   9.6M   1.6G   1% /run
/dev/sda6       233G  218G   3.5G  99% /
tmpfs           7.7G  223M   7.5G   3% /dev/shm
tmpfs           5.0M   4.0K   5.0M   1% /run/lock
tmpfs           7.7G     0  7.7G   0% /sys/fs/cgroup
tmpfs           1.6G   72K   1.6G   1% /run/user/1000
/dev/sda2       195G  147G   49G  76% /mnt
$
```

Descriptori de fișiere

- întregi nenegativi folosiți pentru identificarea fișierelor deschise în sistem
 - alocati de kernel la deschiderea/crearea unui fișier prin program
 - folosiți subsecvent de către program pentru citirea/scrierea fișierului
- descriptori speciali
 - la pornirea oricărui program, shell-ul deschide pentru acesta trei descriptori de fișiere speciali:
 - 0 standard input
 - 1 standard output
 - 2 standard error
 - uzual, asociați cu terminalul de login (sau terminalul de lucru, într-un mediu grafic cu multiple X terminale)
 - `/usr/bin/tty` afișează terminalul asociat unui shell (în general, nu doar terminalul de login)

Redirectarea operatiilor de I/O

- redirectarea operatiilor de I/O se poate face programatic sau direct din shell
- shell-ul intelege constructii sintactice de tipul urmator ca fiind redirectari ale operatiilor de I/O

[n] < filename	redirecteaza citirile de pe descriptorul <i>n</i> catre fisierul desemnat; daca <i>n</i> lipseste, se foloseste <i>stdin</i>
<< marker	redirecteaza <i>stdin</i> catre tastatura folosind un marker de end of file (altfel e Ctrl-d); folosit pentru introducerea documentelor ad-hoc
[n] > filename	redirecteaza scrierile pe descriptorul <i>n</i> catre fisierul desemnat; daca <i>n</i> lipseste, se foloseste <i>stdout</i>
[n] >> filename	adauga scrierile pe descriptorul <i>n</i> la sfarsitul fisierului desemnat ("append"); daca <i>n</i> lipseste, se foloseste <i>stdout</i>

Redirectarea operatiilor de I/O

- ex:

\$ echo "redirectarea stdout in fisierul out" > out

\$ echo "adaugam la sfarsitul fisierului out inca o linie" >> out

\$ cat < out

\$ cat << EOF >> out

> mai adaugam o line la sfarsitul fisierului out

> EOF

\$

Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

Curs 3 - outline

- interpretorul de comenzi
- fisiere si directoare (revizitat)

Interpretorul de comenzi (recapitulare)

- mod de lucru interactiv (comanda-raspuns) sau batch (automatizarea lucrului cu scripturi)
- in mod interactiv, afiseaza un prompt
 - indica faptul ca se asteapta o comanda (interna/externa) de la utilizator
 - uzual, \$ sau % pt utilizatori obisnuiti, # pt root
 - definit de continutul variabilei de mediu PS1

\$ echo \$PS1

- prompt de continuare in variabila de mediu PS2, uzual >

\$ cat << EOF >> out

> mai adaugam o line la sfarsitul fisierului out

> EOF

\$

Mediul de lucru (environment)

- lista de perechi *name = value*
- *name* este numele unei variabile interne a shell-ului
 - Obs: nu orice variabila interna a shell-ului este variabila de mediu
- valorile variabilelor influenteaza comportamentul shell-ului, respectiv al comenzilor externe lansate de acesta
 - setate de SO sau utilizator
- numele scris cu litere mari: PS1, SHELL, HOME, PATH, etc
- valoarea dereferentiata cu ajutorul simbolului \$, eg

\$SHELL = /bin/bash

Variabile de mediu

- setate cu comanda interna *export*
 - marcheaza variabila ca fiind variabila de mediu

\$ export PS1="my-new-prompt> "

- afisate cu comanda */usr/bin/env*
- intr-un program C, accesibile in al treilea parametru al functiei *main* a programului lansat in executie de catre shell

int main(int argc, char argv[], char *envp[])*

The Bourne-Again SHell

- */bin/bash*, urmasul primului shell istoric, */bin/sh* (Bourne Shell)
- fisiere de configurare
 - fisiere de start-up inspectate doar la login
 - system-wide: */etc/profile*
 - locale, in home directory: *~/.profile*, *~/.bash_profile*, *~/.bash_login*
 - continutul lor e executat automat la login
 - fisiere de start-up inspectate la crearea fiecarui terminal (rc file, “run commands”)
 - *~/.bashrc*
 - continutul lor poate fi executat voluntar cu comanda *source* (sau “.”)
source .bashrc
..bashrc
 - fisier de logout: *~/.bash_logout*
 - continutul executat la iesirea din shell (cu *exit* sau Ctrl-d)
 - Obs: Ctrl-d in Unix este caracterul EOF, tiparirea Ctrl-d la prompt termina shell-ul
- istoria comenzilor inregistrata in *~/.bash_history*

Structura comenzilor bash

- pipeline-uri

$\$ cmd_1 | cmd_2 | \dots | cmd_n$ # executie paralela a comenzilor

$\$ cmd_1 | \& cmd_2 | \& \dots | \& cmd_n$ # “|&” e totuna cu “2>&1 |”

- liste de comenzi

$\$ cmd_1; cmd_2; \dots; cmd_n$ # executie secventiala a comenzilor

$\$ cmd_1 \&\& cmd_2 \&\& \dots \&\& cmd_n$

$\$ cmd_1 || cmd_2 \dots || cmd_n$

- variabila bash “?” contine codul de terminare (*exit status*) al ultimei comenzi executate (valoarea zero inseamna succes)

$\$ echo \$?$

Obs: ? nu e variabila de mediu !

Job control

- doua categorii de programe
 - executate in foreground, au acces R/W la terminal
 - executate in background
- comanda incheiata un “&” ruleaza in background
 - shell-ul returneaza imediat utilizatorului promptul

\$ cmd &

- executia unei comenzi in foreground se poate suspenda cu ^Z (Ctrl-z)
 - de fapt, e semnalul SIGTSTP (*kill -SIGTSTP <pid>*)
 - executia comenzii poate fi reluata ulterior, fie in foreground, fie in background
- comanda *jobs* listeaza procesele (joburile) rulate la momentul curent de shell
 - joburile identificate prin numar
 - numarul job-ului poate fi folosit impreuna cu urmatoarele comenzi

<i>\$ kill %n</i>	# termina procesul/job-ul cu nr <i>n</i>
<i>\$ fg %n</i>	# muta in foreground procesul cu nr <i>n</i>
<i>\$ bg %n</i>	# muta jobul <i>n</i> in background
<i>\$ %n &</i>	# muta jobul <i>n</i> in background

Controlul istoriei comenzilor

- `~/.bash_history`

- exemple

`!n` re-executa comanda cu nr *n*

`!-n` re-executa comanda curenta – *n*

`!string` re-executa cea mai recenta comanda care incepe cu *string*

`!?string?` re-executa cea mai recenta comanda care contine *string*

`^str1^str2` repeta comanda anterioara inlocuind *str1* cu *str2*

- interactiv: *Ctrl-r* urmat de un substring al comenzii cautate din istoric

Comenzi

- *interne*: executate direct de catre bash

cd <dir>

alias l='ls -l'

fg/bg/kill <job#>

exit <status> # termina shell-ul cu cod de retur *status*

exec <cmd> # inlocuieste imaginea bash cu imaginea noului proces

de ex: \$ exec firefox

- *externe*: programe de pe disc lansate de catre shell

pwd

echo string

ex escape chars:

echo -e \a # bell

echo -e "aaa\tbbb" # horizontal tab

echo -e "aaa\v\bbbb" # vertical tab + backspace

echo -e "aaa\t\rbbb" # carriage return

echo -e "aaa\t\nbbb" # newline

Tipuri de fisiere

- *fisiere obisnuite (regular files)*: contin date (text sau binare)
- *directoare*: contin numele altor fisiere si informatii despre ele
 - pot fi citite de catre procesele care au permisiunile potrivite
 - DOAR kernelul poate scrie in ele !
- *fisiere speciale, tip device*
 - *caracter*: pt device-uri caracter (ex: tty, seriala)
 - *bloc*: pt device-uri orientate pe bloc (ex: discuri)
 - operatiile de R/W nu se fac prin intermediul FS ci al driverelor

Obs: orice device (echipament) din sistem e fie fisier bloc, fie caracter
- *FIFO: named pipe*, mecanism IPC (Inter-Process Communication)
 - | s.n. *anonymous pipes*, leaga procese *inrudite*
 - conecteaza procese fara legatura
 - fisiere de pe disc, cu nume si politica de acces FIFO
 - bidirectionale, spre deosebire de |

Tipuri de fișiere (cont.)

- *socket*: abstracție pentru IPC peste rețea
 - canal de comunicație local (socket Unix, un fel de FIFO)
 - canal de comunicație între mașini conectate în rețea (socket TCP/IP)
- *link simbolic*: fișier care referă un alt fișier
 - practic fișierul destinație (*link-ul simbolic*) conține numele fișierului sursă (fișierul referit)

\$ ln -s <fișier-sursă> <link-simbolic>

- formatul lung al comenzii *ls* marchează în primul caracter tipul fișierului:

-, d, c, b, p, s, l

- comanda generală, distinge și tipuri de fișiere regulate (text, executabile, imagini, etc):

\$ file <nume-fișier>

Set UID, set GID

- fiecare proces (program in executie) are asociat
 - UID, GID real: identitatea reala a utilizatorului provenita din */etc/passwd*
 - UID, GID efectiv
 - set-UID, set-GID salvate (copii ale UID/GID efectiv)
- in mod normal, UID/GID real = UID/GID efectiv
- cand se executa un program exista posibilitatea de a seta un flag in attributele fisierului executabil a.i.:

“pe durata executiei acestui program UID/GID efectiv al procesului devine UID/GID-ul proprietarului fisierului program”

Ex: comanda de schimbare a parolei utilizator

\$ passwd

/usr/bin/passwd este un program *set-UID* la *root* pt a avea drepturi de scriere in */etc/shadow*

Permisuni de acces la fisiere

- attribute ale fisierului
- grupate in trei categorii
 - permisiuni utilizator: S_IRUSR, S_IWUSR, S_IXUSR
 - permisiuni grup: S_IRGRP, S_IWGRP, S_IXGRP
 - permisiuni pt. alti utilizatori: S_IROTH S_IWOTH, S_IXOTH
 - permisiuni speciale: set-uid, set-gid, sticky bit
- modificabile din shell cu ajutorul comenzii *chmod*

ex: *chmod u+rw <fisier>, chmod g-x <fisier>, chmod o-rwx <fisier>, etc*

sau in octal

chmod 755 <fisier>, chmod 644 <fisier> , etc
- accesul la un fisier: conditionat de combinatia dintre UID efectiv (respectiv GID efectiv) al comenzii executate si bitii de permisiune

umask

- orice fisier nou creat are setata o masca implicita a permisiunilor
 - setata cu comanda *umask* (comanda interna shell)

\$ umask 022

- bitii setati in *umask* sunt off in permisiunile noului fisier creat
- de regula, masca setata in fisierele de configurare shell (eg, */etc/profile*)

Stergerea fisierelor

- un fisier poate avea m.m. *link-uri* la aceeași structura internă din kernel (*i-node*)
 - nume diferite ale aceluiași fisier
 - s.n. *link-uri hard*
 - create cu comanda *ln*

\$ ln <fisier-sursa> <fisier-destinatie>

- stergerea unui link nu înseamnă stergerea fisierului de pe disc !
 - stergerea ultimului link șterge și fisierul
- pt. a șterge o intrare de fisier dintr-un director se folosește *rm*

\$ rm <nume-fisier>

- stergerea necesită două permisiuni:
 - permisiunea de a scrie în director
 - permisiunea de a căuta în director (bitul *x* de execuție setat în directorul din care ștergem fisierul)

Link-uri simbolice

- limitari *link-uri hard*
 - link-ul si fisierul linkat trebuie sa se afle pe acelasi sistem de fisiere (disc formatat)
 - doar *root-ul* poate crea linkuri hard catre directoare
- *link symbolic*: fisier care contine numele fisierului referit (un string)
- utilizate pentru a circumventa limitarile link-urilor hard
- create cu comanda *ln* si flag-ul *-s*

```
$ ln -s /etc/profile ~/.system-wide-profile
```

- in general, comenzile shell dereferentiaza linkul simbolic
 - exceptii: *lstat*, *remove*, *rename*, *unlink*, *samd*
 - ex: pt. linkul simbolic creat mai sus

```
$ rm ~/.system-wide-profile    # sterge link-ul simbolic, nu sursa, adica /etc/profile
$ ls -l ~/.system-wide-profile # afiseaza attributele linkului symbolic, nu ale sursei
$ cat ~/.system-wide-profile   # afiseaza continutul /etc/profile
```

Lucrul cu directoare

- create cu *mkdir*, sterse cu *rmdir*

```
$ mkdir <director>
```

```
$ mkdir -p </director>/<subdirector> # creeaza si directoarele  
# inexistente on the fly
```

```
$ rmdir <director>
```

- Obs: *rmdir* nu poate sterge un director decat daca e GOL !
- schimbarea directorului curent

```
$ cd <director> # schimba directorul in <director>
```

```
$ cd # ⇔ cd $HOME
```

```
$ cd - # schimba directorul curent in directorul anterior
```

- aflarea directorului de lucru curent (current working directory)

```
$ pwd
```

Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

Curs 4 - outline

- fisiere si directoare (epilog)
- procese
- inter-process communication
- semnale

Sumar comenzi utile pt. lucrul cu fisiere

- *mkdir* – creeaza directoare
- *rmdir* – sterge directoare (cu conditia sa fie goale)
- *touch* – creeaza un fisier gol daca nu exista deja
- *mv* – muta directoare sau fisiere
 - redenumeste, nu se face copiere fizica decat daca datele se muta de pe un disc pe altul
- *cp* – copiaza (fizic, duplica) directoare (cu *-r*) sau fisiere
 - se pot copia directoare/fisiere multiple intr-un director destinatie
\$ cp <fisier-1> <fisier-2> <fisier-n> <director>
- *rm* – sterge directoare (cu *-r*) sau fisiere
- *ln* – creeaza linkuri hard sau simbolice
- *mknod* – creeaza fisiere speciale tip caracter, bloc sau FIFO

\$ mknod /dev/sda3 b 8 3

Wildcards

- caractere speciale interpretate de shell
- `^` – simbolizeaza inceputul liniei
- `$` – simbolizeaza sfarsitul liniei
- `atom` = caracter sau set de caractere grupat cu `[]` sau `()`
- `*` – 0 sau mai multe aparitii ale atomului precedent
- `+` – cel putin 1 aparitie a atomului precedent, posibil mai multe
- `?` – cel mult o aparitie a atomului precedent (0 sau 1 aparitii)

Cautarea in fisiere

- comanda uzuala: *grep*
- foloseste tipare si expresii regulate
- synopsis
\$ grep <expresie> <fisiere>
- optiuni utile
 - R / -r cautare recursiva (cu/fara dereferentiere linkuri simbolice)
 - i case insensitive
 - n tipareste nr liniei pe care s-a gasit expresia
 - w cauta cu exactitate tiparul furnizat (cuvant, word)
 - v inverseaza sensul matching-ului (afiseaza liniile care nu se potrivesc)
- ex:
*\$ grep printf *.c*
*\$ grep -v -w printf *.ch*
\$ ls -l | grep ^d

Cautarea in directoare

- comanda uzuala: *find*

- synopsis:

\$ find <pathname> -name <expresie>

- optiuni utile

-exec executa comanda specificata asupra elementelor gasite

-type limiteaza rezultatele afisate la un anumit tip (eg, fisiere)

-iname similar cu *-name* dar case insensitive

-maxdepth limiteaza nivelul de recursivitate

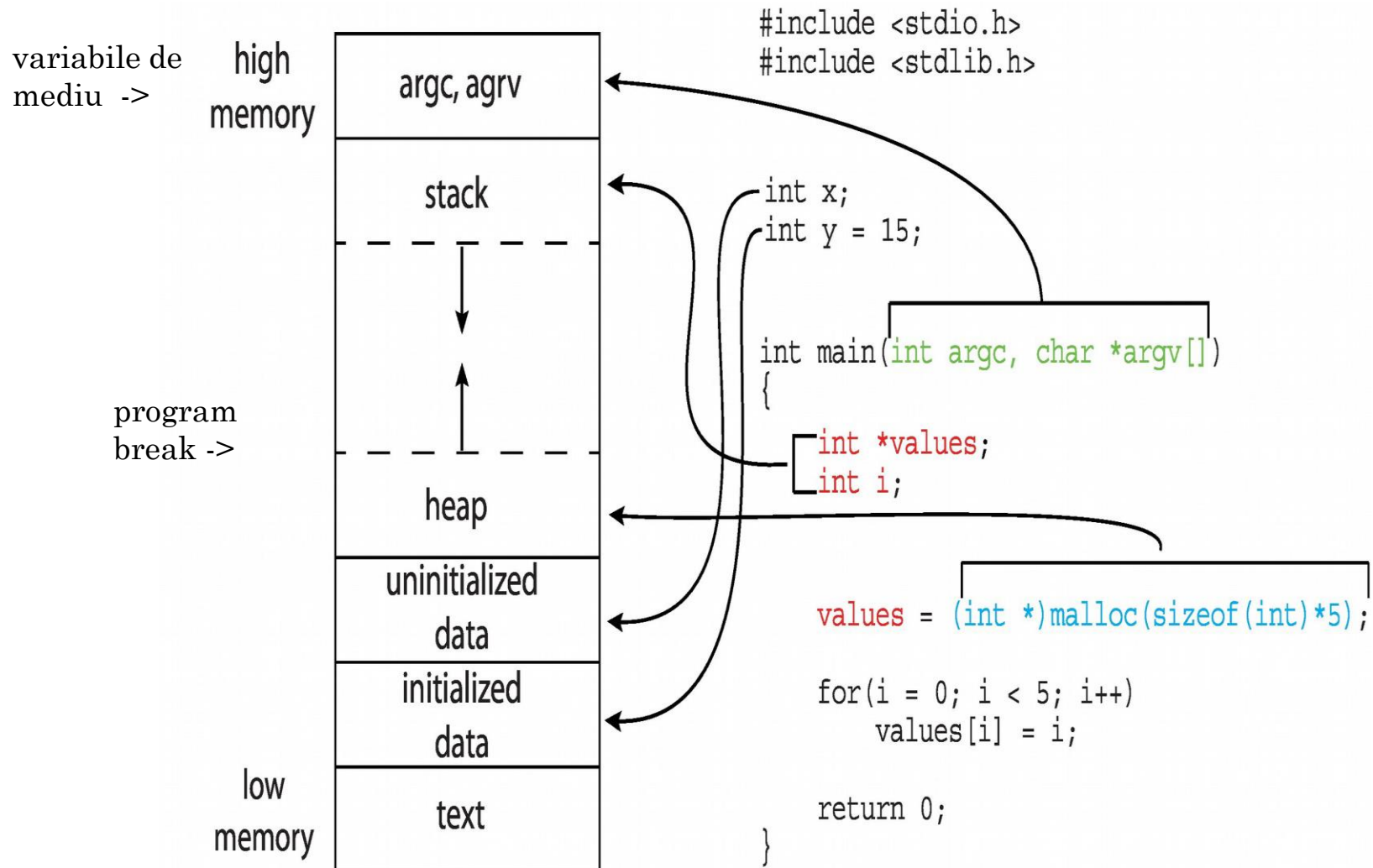
- ex: *\$ find / -name hello*

*\$ find . -iname *.o -exec rm {} \;*

Procese

- proces = abstractia executiei unui program (instr + date)
 - in fapt, este programul + starea executiei sale la un moment dat (reflectata de registrele CPU si valorile variabilelor din program)
 - identificat prin *Process ID (PID)*
 - comanda *ps* afiseaza PID-urilor proceselor aflate in rulare momentan
 - mai multe instante in rulare ale aceluasi program sunt procese diferite, cu PID-uri diferite
- executia proceselor este secventiala, nu exista executie paralela a instructiunilor intr-un singur proces
- mai multe parti
 - codul program, cunoscut si ca sectiunea de *text*
 - starea curenta reflectata de registrele CPU
 - *stiva* contine date temporare
 - parametrii functiilor, adrese de retur, variabile locale
 - sectiunile de *date initializate/neinitializate*
 - contin datele globale
 - *heap-ul* contine memoria alocata dinamic de catre program in timpul executiei

Imaginea unui program C in memorie

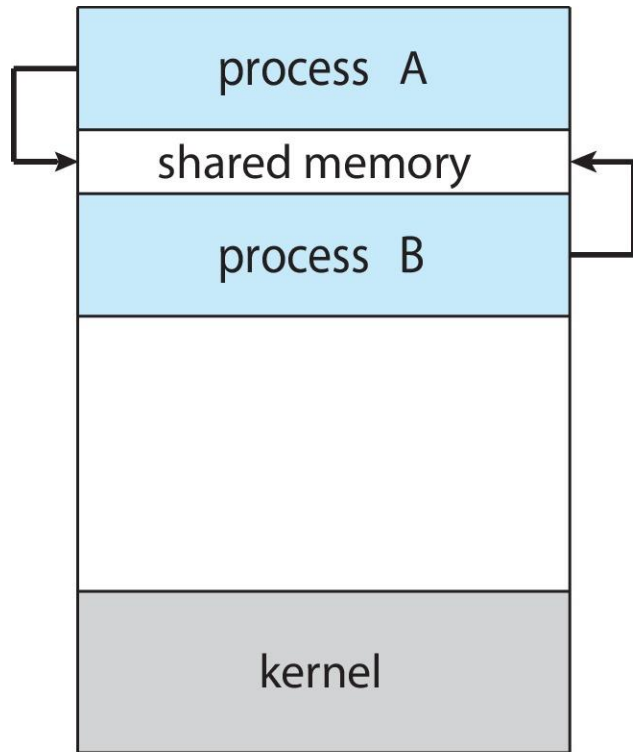


Comunicare inter-procese (IPC)

- procesele pot fi *independente* sau *cooperante*
- procesele cooperante pot afecta sau pot fi afectate de alte procese, inclusiv prin partajarea datelor
- motive de cooperare
 - partajarea informatiei
 - accelerarea calculului
 - modularitate
 - confort
- procesele cooperante necesita mijloace de comunicare inter-proces (IPC)
- modele IPC
 - memorie partajata
 - schimb de mesaje (message passing)

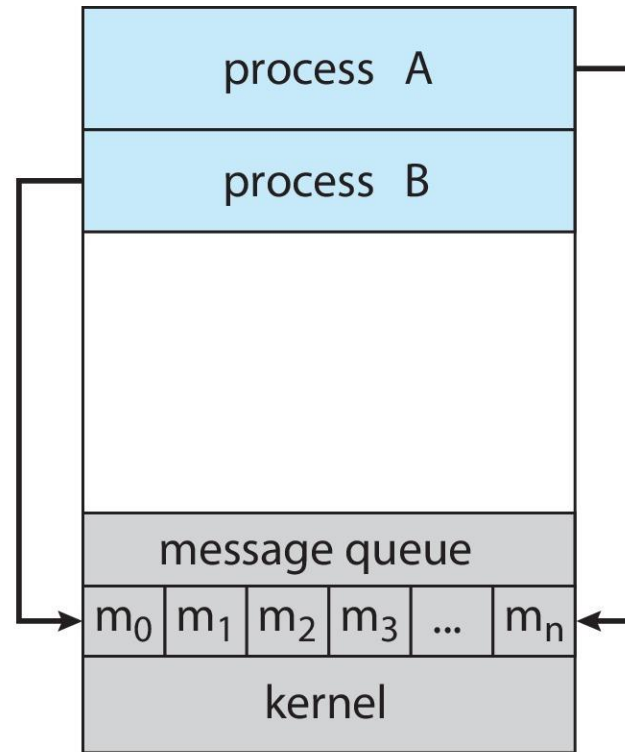
Modele de comunicare

(a) Memorie partajata.



(a)

(b) Message passing.



(b)

IPC – Message Passing

- procesele comunica intre ele fara variabile partajate
- operatii:
 - **send**(*message*)
 - **receive**(*message*)
- dimensiunea mesajului este fixa sau variabila

Tipuri de comunicare

- *directa*: procesele isi folosesc identitatea explicit
 - **send** ($P, message$) – trimite un mesaj procesului P
 - **receive**($Q, message$) – primeste un mesaj de la procesul Q
 - ex: *sockets* (canale de comunicatie TCP/IP)
- *indirecta*: mesajele sunt trimise si receptionate in/din casute postale (numite si porturi)
 - fiecare casuta are un ID unic
 - procesele pot comunica doar daca partajeaza o casuta
 - operatii
 - creeaza casuta (port)
 - trimite si primeste mesaje prin intermediul casutei postale
 - sterge casuta
 - primitive de comunicare
 - **send**($A, message$) – trimite mesaj in casuta postala A
 - **receive**($A, message$) – primeste mesaj din casuta postala A

Sincronizare

- schimbul de mesaje poate fi **blocant** sau **neblocant**
- **modul blocant** este considerat **sincron**
 - **send blocant** – transmitatorul e blocat pana cand se primeste mesajul
 - **receive blocant** – receptorul e blocat pana cand un mesaj e disponibil
- **modul neblocant** este considerat **asincron**
 - **send neblocant** – transmitatorul trimite mesajul si continua
 - **receive neblocant** – receptorul primeste:
 - un mesaj valid, sau
 - mesaj vid (null)
- sunt posibile diferite combinatii
 - daca *send* si *receive* sunt ambele blocante, avem un **rendezvous**

Comunicare FIFO in Unix

- prin *pipe-uri* anonime sau cu nume
- *pipe-uri anonime*
 - combinatie de comunicare directa/indirecta
 - procesele cooperante isi cunosc implicit identitatea
 - comunicare unidirectionala
\$ cat hello.c | grep printf
- *pipe-uri cu nume (FIFOs)*
 - fisiere speciale cu nume
 - comunicare indirecta, procesele cooperante nu isi cunosc identitatea
 - create cu comenzile *mknod/mkfifo*
 - odata create, folosite la fel ca fisierele, dar cu politica de acces FIFO

```
$ mknod myfifo p          # ⇔ mkfifo myfifo
$ cat < myfifo &          # trebuie sa existe mai intai un
                           # receptor care asteapta date
$ echo "write some message in myfifo" > myfifo
```

Semnale

- notificari asincrone ale procesului referitoare la producerea anumitor evenimente
 - echivalentul software al exceptiilor (HW sau SW)
 - se trimit fie intre procese, fie de catre kernel catre un proces
 - generate din mai multe surse:
 - apel sistem *kill(pid, semnal)*
 - comanda *kill*: `kill -TERM <pid>`
 - tastatura: DEL, Ctrl-C, Ctrl-Z, etc
 - anumite evenimente soft si hard generate de kernel
 - ex:
 - terminarea unui proces notificata asincron parintelui cu SIGCHLD
 - procesul care executa o impartire la zero primeste un semnal SIGFPE
 - accesul ilegal la memorie (eg, memorie nealocata) genereaza SIGSEGV
- reactia procesului la primirea unui semnal depinde de
 - tipul semnalului
 - decizia programului de a trata sau nu evenimentul semnalat

Posibilitati de tratare a semnalelor

1) semnalul e ignorat

- specific pt. evenimente care nu rezulta in erori/consecinte majore

2) terminarea programului (semnalul “ucide” procesul)

- valabil pt. restul evenimentelor

3) tratarea semnalului cf. indicatiei/dispozitiei programului, despre care se spune ca “prinde” semnalul

- se face cu ajutorul unei rutine de tratare a semnalului (handler)

Obs: nu orice semnal poate fi “prins” !

Ex: SIGKILL/SIGSTOP nu pot fi prinse, KILL termina invariabil programul

\$ kill -KILL <pid>

⇔ kill -9 <pid>

Exemple utilizare semnale

- comanda *kill*

- foloseste un PID identificat in prealabil cu comanda *ps*

\$ kill -<nume-semnal/nr-semnal> <PID>

\$ kill -TERM 4899 *# ⇔ kill 4899 sau kill -15 4899*

Obs: numele semnalului poate fi complet, eg, SIGTERM sau prescurtat, eg TERM

\$ kill -l *# afiseaza toate semnalele disponibile*

\$ kill -l 15 *# mapeaza nr de semnal in nume*

- semnalele pot fi generate voluntar de catre utilizator

- ex: *Ctrl-c* genereaza SIGINT

- uzual termina procesul rulat de shell

- daca procesul prinde SIGINT, la apasarea *Ctrl-c* se executa signal handlerul asociat SIGINT de catre proces (i.e., procesul nu moare automat)

\$ kill -SIGINT 4899 *# ⇔ kill -2 4899*

- *Ctrl-z* genereaza SIGTSTP

- suspenda executia procesului curent (poate fi continuat cu comenzi tip job control, *fg/bg*)

\$ kill -SIGTSTP 4899 *# ⇔ kill -20 4899*

Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

Curs 5 - outline

- programare shell
- scripturi
- variabile
- teste
- instructiuni conditionale
- instructiuni iterative si repetitive
- functii

Motivatie

- de multe ori vrem să avem o singură comandă pentru un sir de comenzi
- comenzile sunt preponderent comenzi shell sau din sistem, nu operatii logice sau aritmetice
- dezvoltare rapida pentru programe scurte (in general)
 - in particular, folosite extensiv de sistemele de operare la pornire/oprire (eg, pt servicii) & administrare
- portabilitate: vrem să functioneze pe si pe alte sisteme pt. automatiza operatii de configurare sau administrare
- surse: tutoriale internet (eg, CS2043 Unix and Scripting de la Cornell University)

Variante Shell

- sh(1) – Bourne shell
 - printre primele shelluri
 - disponibilă oriunde în /bin/sh
 - functionalitate redusă
 - portabilitate
- csh(1) – C shell, comenzi apropiate de modul de lucru în C
- ksh(1) – Korn shell, bazat pe sh(1), succesori csh(1)
- bash(1) – Bourne Again shell
 - cea mai răspândită
 - există și în Windows 10
 - majoritatea scripturilor moderne scrise în bash(1)
 - acest rezultat a dus la o lipsă de portabilitate
 - proiect mare cu multe linii de cod și probleme de securitate

Variabile Shell

- două tipuri de variabile: locale și de mediu
- variabilele locale există doar în instanța curentă a shell-ului
- convenție: variabilele locale se notează cu litere mici
- asignare: `x=1` (fără spații!)
- `x = 1`: se interpretează drept comanda `x` cu argumentele `=` și `1`
- folosirea valorii unei variabile se folosește prin prefixarea numelui cu `$`
- conținutul oricărei variabile poate fi afișat cu comanda `echo(1)`

```
$ x=1
$ echo $x
1
```

Variabile de mediu

- folosite de sistem pentru a defini modul de functionare a programelor
- shell-ul trimite variabilele de mediu proceselor sale copil
- `env(1)` – afisează toate variabilele de mediu setate
- variabile importante
 - `$HOME` – directorul în care se țin datele utilizatorului curent
 - `$PATH` – listă cu directoare în care se caută executabilele
 - `$SHELL` – programul de shell folosit implicit
 - `$EDITOR` – programul de editare fișiere implicit
 - `$LANG` – limba implicită (ex. `ro_RO.UTF-8`)
- `export(1)` – se folosește pentru a seta o variabilă de mediu
- `printenv(1)` – se folosește pentru a afișa o variabilă de mediu

```
$ export V=2
$ printenv V
2
$ echo $V
2
```

Variabile locale vs mediu

Variabila locala

```
$ V = 2
```

```
$ echo $V
```

```
2
```

```
$ b a s h
```

```
bash$ echo $V
```

```
bash $
```

Variabila de mediu

```
$ V = 2
```

```
$ echo $V
```

```
2
```

```
$ export V
```

```
$ b a s h
```

```
bash$ echo $V
```

```
2
```

Variable expansion

Variabilele pot fi mai mult decât simpli scalari

- `$(cmd)` – evaluează întâi comanda `cmd`, iar rezultatul devine valoarea variabilei

```
$ echo $(pwd)
/home/paul
$ x=$(find . -name \*.c)
$ echo $x
./batleft.c ./pcie.c ./maxint.c ./eisaidd.c
```

- `$((expr))` – evaluează întâi expresia aritmetică + side effects

\$ x=1	\$ echo \$((x++))
\$ echo \$((1+1))	1
2	\$ echo \$((x++))
\$ echo \$((x+1))	2
2	\$ echo \$((++x))
\$ echo \$((x<1))	4
0	

Quoting

Sirurile de caractere sunt interpretate diferit în funcție de citare:

- Single quotes `' '`
 - toate caracterele își păstrează valoarea
 - caracterul `'` nu poate apărea în sir, nici precedat de `"\"`
 - exemplu:
 - `$ echo 'Am o variabila $x'`
`Am o variabila $x`
- Double quotes `" "`
 - caractere speciale `$ ' \"` (optional `!`)
 - restul caracterelor își păstrează valoarea
 - exemplu:
 - `$ echo "$USER has home in $HOME"`
`paul has home in /home/paul`

Quoting (cont.)

Sirurile de caractere sunt interpretate diferit în funcție de citare:

- Back quotes ` – funcționează ca `$()`

```
$ echo "Today is `date`"
```

```
Today is Wed May  2 18:00:08 EEST 2018
```

Înlănțuirea comenzilor

- `cmd1; cmd2` – înlănțuire secvențială, `cmd2` imediat după `cmd1`
- `cmd1 | filtru | cmd2` – ieșirea comenzii din stânga este intrarea celei din dreapta operatorului `|`
- `cmd1 && cmd2` – execută a doua comandă doar dacă prima s-a executat cu succes
- `cmd1 || cmd2` – execută a doua comandă doar dacă prima a eșuat
- exemplu:

```
$ mkdir acte && mv *.docx acte/
```

```
$ ls -lR | tee files.lst | wc -l
```

```
$ ssh example.org || echo "Connection failed!"
```

Scripting

Script = program scris pentru un mediu run-time specific care automatizează executia comenzilor ce ar putea fi executate alternativ manual de către un operator uman.

- nu necesită compilare
- executia se face direct din codul sursă
- de acea programele ce execută scriptul se numesc interpretoare în loc de compilatoare
- exemple: perl, ruby, python, sed, awk, ksh, csh, bash

Indicii de interpretare

- semnalate cu ajutorul string-ului `#!` pe prima linie a scriptului
 - are forma `#!/path/to/interpreter`
- exemple: `#!/bin/sh`, `#!/usr/bin/python`
- pentru portabilitate folositi `env(1)` pentru a găsi unde este instalat interpretorul
- exemple: `#!/usr/bin/env ruby`, `#!/usr/bin/env perl`
- oriunde altundeva în script `#` este interpretat ca început de comentariu și restul linei este ignorată (echivalent `//` în C)
- introdusă de Denis Ritchie circa 1979

Exemplu: hello.sh

1. Scriem fisierul hello.sh cu comenzile dorite

```
#!/bin/sh
```

```
# Salute the user  
echo "Hello , $USER!"
```

2. Permite executia: `chmod +x hello.sh`

3. Executăm:

```
$ ./hello.sh
```

```
Hello , paul!
```

Alternativ:

```
$ sh hello.sh # nu necesita permisiunea de executie
```

Variabile speciale în scripturi

- `$1, $2, ..., ${10}, ${11}, ...` – argumentele în ordinea primită
- dacă numărul argumentului are mai mult de două cifre, trebuie pus între acolade
- `$0` – numele scriptului (ex. `hello.sh`)
- `$#` – numărul de argumente primite
- `$*` – toate argumentele scrise ca `"$1 $2 ... $n"`
- `$@` – toate argumentele scrise ca `"$1" "$2" ... "$n"`
- `$?` – valoarea ieșirii ultimei comenzi executate
- `$$` – ID-ul procesului curent
- `$_` – ID-ul ultimului proces suspendat din execuție
- shiftarea parametrilor pozitionali la stanga: comanda *shift*

Example: arguments script

- add.sh – adună două numere

```
#!/bin/sh
```

```
echo $(( $1 + $2 ))
```

apel:

```
$ sh add.sh 1 2
```

```
3
```

Varianta add2.sh:

- ```
#!/bin/sh
```

```
sum = $1
```

```
shift
```

```
echo $(($sum + $1))
```



# Example: argumente script

- tolower.sh – imită funcția din C tolower(3)

```
#!/bin/sh
```

```
tr '[A-Z]' '[a-z]' < $1 > $2
```

apel:

```
$ echo "WHERE ARE YOU?" > screaming.txt
```

```
$./tolower.sh screaming.txt decent.txt
```

```
$ cat decent.txt
```

```
where are you?
```

# Blocuri de control

Pentru scripturi mai complexe avem nevoie, ca în orice limbaj, de blocuri de control

- conditionale – `if`, `test [ ]`, `case`
- iterative – `for`, `while`, `until`
- comparative – `-ne`, `-lt`, `-gt`
- functii – `function`
- iesiri – `break`, `continue`, `return`, `exit`

# If

- cuvinte cheie: if, then, elif, else, fi

- ex:

```
if test-cmd
```

```
then
```

```
 cmds
```

```
elif test-cmd
```

```
then
```

```
 cmds
```

```
else
```

```
 cmds
```

```
fi
```

- rezultatul *test-cmd* apare in \$? (0 -> true, !=0 ->false)

# Exemplu: if

Caută în fisier date și le adaugă dacă nu le găsește

```
#!/bin/sh
if grep "$1" $2 > /dev/null
then
 echo "$1 found in file $2"
else
 echo "$1 not found in file $2, appending"
 echo "$1" >> $2
fi
```

apel:

```
$./text.sh who decent.txt
who not found in file decent.txt , appending
$ cat decent.txt
where are you?
who
```

# test sau [ ]

- nu dorim să testăm tot timpul executia unei comenzi
- există expresii pentru a compara sau verifica variabile
- `test expr` – evaluează valoarea de adevăr a lui `expr`
- `[ expr ]` – efectuează aceiasi operatie (**atentie la spatii!**)
  - `[` este comanda interna
  - `$ type [`
  - `[` is a shell builtin
- expresiile pot fi legate logic prin
  - `[ expr1 -a expr2 ]` – conjunctie
  - `[ expr1 -o expr2 ]` – disjunctie
  - `[ ! expr ]` – negatie

# Expresii test: numere

- [ n1 -eq n2 ] –  $n_1 = n_2$
- [ n1 -ne n2 ] –  $n_1 \neq n_2$
- [ n1 -ge n2 ] –  $n_1 \geq n_2$
- [ n1 -gt n2 ] –  $n_1 > n_2$
- [ n1 -le n2 ] –  $n_1 \leq n_2$
- [ n1 -lt n2 ] –  $n_1 < n_2$

# Expresii test: siruri de caractere

- `[ str ]` – str are lungime diferită de 0
- `[ -n str ]` – str nu e gol
- `[ -z str ]` – str e gol ("" )
- `[ str1 = str2 ]` – stringuri identice
- `[ str1 == str2 ]` – stringuri identice
- `[ str1 != str2 ]` – stringuri diferite

# Expresii test: fișiere

- `-e path` – verifică dacă există calea `path`
- `-f path` – verifică dacă `path` este un fișier obisnuit
- `-d path` – verifică dacă `path` este un director
- `-r path` – verifică dacă aveți permisiunea de a citi `path`
- `-w path` – verifică dacă aveți permisiunea de a scrie `path`
- `-x path` – verifică dacă aveți permisiunea de a executa `path`



# while

- execută blocul cât timp comanda *cmd* se execută cu succes

```
while cmd
```

```
do
```

```
 cmd1
```

```
 cmd2
```

```
done
```

- în loc de comandă putem avea o expresie de test
- într-o singură linie: `while cmd; do cmd1; cmd2; done`

# Exemplu: while

Afisează toate numerele de la 1 la 10:

```
i=1
while [$i -le 10]
do
 echo "$i"
 i=$(($i+1))
done
```

Sau într-o singură linie:

```
i=1; while [$i -le 10]; do echo "$i"; i=$(($i+1));
done
```

# until

- execută blocul cât timp comanda *cmd* se execută **fără** succes
- `until cmd`  
do  
    `cmd1`  
    `cmd2`  
done
- în loc de comandă putem avea o expresie de test
- într-o singură linie: `until cmd; do cmd1 ; cmd2; done`

# for

- execută blocul pentru fiecare valoare din listă

```
for var in str1 str2 ... strN
do
 cmd1
 cmd2
 ...
done
```

- var ia pe rând valoarea str<sub>1</sub>, pe urmă str<sub>2</sub> până la str<sub>N</sub>
- de regulă comenzile din bloc (cmd<sub>1</sub>, cmd<sub>2</sub>) sunt legate de *var*
- comanda for are mai multe forme, aceasta este cea mai întâlnită
- într-o singură linie:  
for var in str<sub>1</sub> str<sub>2</sub> ... str<sub>N</sub>; do cmd<sub>1</sub> cmd<sub>2</sub>; done

# Exemplu: for

Compilează toate fişierele C din directorul curent

```
for f in *.c
do
 echo "$f"
 cc $f -o $f.out
done
```

Sau într-o singură linie:

```
for f in *.c; do echo "$f"; cc $f -o $f.out; done
```

# for traditional

- formă întâlnite doar în unele shell-uri, **nu este portabil**
- execută blocul urmând o sintaxă similară C

```
for ((i=1; i<=10; i++))
do
 echo $i
done
```

- `i` ia pe rând toate valorile între 1 și 10
- în exemplu, blocul afisează `$i`, dar pot apărea oricâte alte comenzi
- într-o singură linie:

```
for ((i=1; i<=10; i++)); do cmd1; cmd2; done
```

# Case

- se comportă similar cu switch în C

```
case var in
 pattern1)
 cmds
 ;;
 pattern2)
 cmds
 ;;
 *)
 defaultcmd
 ;;
esac
```

- pattern – string sau orice expresie de shell (similar cu wildcards-urile folosite pentru grep(1))

# Exemplu: case

```
echo "Greetings !"
read input_string
case $input_string in
 hi)
 echo "A good day to you too !"
 ;;
 salut)
 echo "Buna ziua !"
 echo "Ou, peut-etre, bonjour a vous aussi ? "
 ;;
 moin)
 echo "Guten morgen !"
 ;;
 *)
 echo "Sorry, I don't understand !"
 ;;
esac
```



# Funcții

- întorc valori:
  - schimbând valoarea variabilelor
  - folosind comanda *exit* pentru a termina scriptul
  - folosind comanda *return* pentru a termina funcția
  - tipărind rezultate la *stdout*
    - recuperate de apelant în maniera tipică shell-urilor

e.g., `var=`expr ...``
- nu pot modifica parametrii de apel, dar pot modifica parametrii globali

# Exemplu functie

```
#!/bin/sh
```

```
add_host()
{
 IP_ADDR=$1
 HOSTNAME=$2
 shift; shift;
 ALIASES=$@
 echo "$IP_ADDR \t $HOSTNAME \t $ALIASES" >> $hostfile
}
```

```
aici incepe scriptul
hostfile=$1
echo "Start script"
add_host 80.96.21.88 fmi.unibuc.ro fmi
add_host 80.96.21.209 www.unibuc.ro www
```

Obs: *add\_host* parsata de shell si verificata sintaxa, dar executata doar la apelare

# Scope-ul variabilelor

- cu exceptia parametrilor, nu exista scope pt variabile

e.g. *scope.sh*

```
#!/bin/sh
```

```
somefn()
```

```
{
```

```
 echo "Function call parameters are $@"
```

```
 a=10
```

```
}
```

```
echo "script arguments are $@"
```

```
a=11
```

```
echo "a is $a"
```

```
somefn first string next
```

```
echo "a is $a"
```

```
Apel:
```

```
$./scope.sh 1 2 3
```

```
script arguments are 1 2 3
```

```
a is 11
```

```
Function call parameters are first string next
```

```
a is 10
```

# Recursivitate

- factorial.sh:

```
#!/bin/sh
```

```
factorial()
```

```
{
```

```
 if ["$1" -gt "1"]; then
```

```
 prev=`expr $1 - 1`
```

```
 rec=`factorial $prev`
```

```
 val=`expr $1 * $rec`
```

```
 echo $val
```

```
 else
```

```
 echo 1
```

```
 fi
```

```
}
```

```
echo -n "Input some number: "
```

```
read n
```

```
factorial $n
```

# Biblioteci de functii

- colectie de functii grupate intr-un fisier CARE NU INCEPE cu linia speciala `#!` !
- se pot defini variabile globale
- “apelul” propriu-zis se face folosind comanda *source*

e.g., *renamelib.sh*:

```
MSG="Renaming files ..."
```

```
rename()
```

```
{
```

```
 mv $1 $2
```

```
}
```

Apelul functiei de biblioteca in scriptul *libcall.sh*:

```
#!/bin/sh
```

```
./renamelib.sh # echivalent cu "source ./renamelib.sh"
```

```
echo $MSG
```

```
rename $1 $2
```

# Coduri de retur

```
#!/bin/sh
add_host()
{
 if ["$#" -eq "0"]; then
 return 1
 fi
 IP_ADDR=$1
 HOSTNAME=$2
 shift; shift;
 ALIASES=$@
 echo "$IP_ADDR \t $HOSTNAME \t $ALIASES" >> $hostfile
}
hostfile=$1
echo "Start script"
add_host
RET_CODE=$?
if ["$RET_CODE" -eq "1"]; then
 echo "No arguments to the add_host call !"
fi
```

# read

- citește una sau mai multe variabile din stdin
- sintaxă: `read var1 var2 ... varN`

```
$ read x y
```

```
1 2
```

```
$ echo $x $y
```

```
1 2
```

- fără nici o variabilă, pune tot rezultatul în \$REPLY

```
$ read
```

```
hello
```

```
$ echo $REPLY
```

```
hello
```

- citire linie cu linie dintr-un fișier:  
`cat foo.txt | while read LINE; do echo $LINE; done`

# Depanare

Pentru a depana un script apelati-l cu shell-ul folosit si optiunea -x.  
Comenzile executate apar pe ecran prefixate cu + .

```
$ sh -x tolower.sh screaming.txt decent.txt
+ tr [A-Z] [a-z]
+ < screaming.txt
+ > decent.txt
```



# Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

# Curs 7 - outline

- filtre
- editarea automata a textelor
- procesarea automata a textelor

# Procesare text: cmd1 | filtru | cmd2

## Vocabular

- filtru – program, comandă, operație care procesează ieșirea în format text a unei comenzi astfel încât noua formă să poată fi procesată mai departe de utilizator sau de un alt program
- linie – sir de caractere care se termină cu `\n` (sau `\r\n` în Windows)
- separator (sau delimitator) – caracter sau sir folosit pentru a delimita datele într-o linie
- câmp (*field*) – un subsir dintr-o linie care reprezintă un anumit tip de date
- exemplu:
  - linie: nume prenume grupă serie
  - separator: " "
  - câmpuri: nume, prenume, grupă și serie

# cut(1)

cut(1) extrage câmpuri din fiecare linie primită la intrare

- `list` – numere sau intervale separate de virgulă sau spații
- `-b list` – lista conține poziții în bytes
- `-c list` – lista conține pozițiile caracterelor
- `-f list` – lista specifică câmpuri
- `-d delim` – specifică delimitatorul pentru câmpuri (implicit este `\t`)
- `-n` – nu împarte caractere multi-byte în bytes
- `-s` – sare peste liniile care nu contin delimitatoare

Tipuri de apel

- `cut -b [-n] list [file ...]`
- `cut -c list [file ...]`
- `cut -f list [-s] [-d delim] [file ...]`

# Example: cut(1)

Afisează numele și shellurile folosite de utilizatorii din sistem:

```
$ cut -d : -f 1,7 /etc/passwd
nobody:/sbin/nologin
paul:/bin/ksh
build:/bin/ksh
joe:/bin/ksh
_mysql:/sbin/nologin
_postgresql:/bin/sh
souser:/bin/ksh
alex:/bin/ksh
```

# Example: cut(1)

Arată numele si data la care s-au logat utilizatorii activi:

```
$ who | cut -c 1-8,18-30
```

|          |     |    |       |
|----------|-----|----|-------|
| deraadt  | May | 8  | 18:32 |
| dlg      | May | 3  | 20:39 |
| jsing    | Apr | 28 | 06:47 |
| landry   | Apr | 19 | 14:22 |
| deraadt  | Apr | 19 | 08:24 |
| kettenis | May | 9  | 02:47 |
| deraadt  | May | 3  | 22:18 |
| pirofti  | May | 9  | 04:36 |

# paste(1)

Lipește fișierele primite la intrare pe coloane (pe verticală)

- `-d list` – folosește delimitatorul pentru a înlocui caracterul linie nouă `\n` din fiecare fișier
- `-s` – serializare
- `-` – reprezintă intrarea standard (stdin)

Apel

- `paste [-s] [-d list] file ...`

# Exemplu: paste(1)

Fie fisierele cu nume și prenume:

```
$ echo "Paul\nAlex\nAna" > firstnames.txt
$ echo "Irofti\nAlexandrescu\nPopescu" \
 > lastnames.txt
```

Implicit, paste(1) lipește numele de prenume

```
$ paste firstnames.txt lastnames.txt
Paul Irofti
Alex Alexandrescu
Ana Popescu
```

Acelasi lucru dar cu serializare

```
$ paste -s firstnames.txt lastnames.txt
Paul Alex Ana
Irofti Alexandrescu Popescu
```



# Exemplu: paste(1)

Afisează fișierele din directorul curent pe trei coloane:

```
$ ls | paste - - -
1 10 11
2 3 4
5 6 7
8 9 intro
```

Identical cu apelul `ls | paste -s -d '\t\t\n' -`

Crează o listă de directoare bin din sistem separate prin :

```
$ find / -name bin -type d | paste -s -d : -
/usr/X11R6/bin:/usr/local/bin:
/usr/local/lib/qt4/bin:
...
```

# split(1)

Împarte fisierul dat în mai multe fișiere de 1000 de linii fiecare.

- `-a suffix_length` – câte litere să contină sufixul noilor fișiere
- `-b byte_count` – crează fișiere de lungimea dată în bytes
- `-l line_count` – crează fișiere cu numărul de linii dat
- `file` – fisierul de împărțit, implicit este `stdin`
- `name` – prefixul pentru noile fișiere

Apel

- `split [options] [file [name]]`

Implicit crează fișierele `xaa`, `xab`, `xac`, ...

# Exemplu: split(1)

- Împarte fisierul LaTeX în mai multe fișiere de 100 de linii:

```
$ wc -l fisier.tex
 362 fisier.tex
$ split -l 100 fisier.tex
$ ls x*
xaa xab xac xad
$ wc -l x*
 100 xaa
 100 xab
 100 xac
 62 xad
 362 total
```

Acelasi lucru dar cu prefix fisier și o singură literă sufix:

```
$ split -a 1 -l 100 fisier.tex fisier
```

# join(1)

Alăturează linii care contin chei identice din două fisiere diferite

- `file1`, `file2` – fisierele de intrare
- `-1 field` – câmpul cheie din fisierul 1
- `-2 field` – câmpul cheie din fisierul 2
- `-a file_number` – produce o linie pentru fiecare nepotrivire din fisierul dat (1 sau 2)

Apel

- `join [-1 field] [-2 field] [options] file1 file2`

# Exemplu: join(1)

Alătură persoanele pentru care exista date legate de vârstă și venit:

```
$ cat age.txt
```

```
Paul 33
```

```
Alex 40
```

```
Ana 25
```

```
$ cat income.txt
```

```
Paul 3000
```

```
Ana 4500
```

```
$ join age.txt income.txt
```

```
Paul 33 3000
```

```
Ana 25 4500
```

Acelasi lucru dar include si persoanele (ex. Alex) fără venit:

```
$ join -a1 age.txt income.txt
```

# Basic Calculator – bc(1)

Calculator pentru operatii aritmetice și logice

- -l – permite operatii cu numere în virgulă mobilă
- -e expr – evaluează expresia, pot fi mai multe
- file – preia expresii din fisier
- operatorii binari sunt la fel ca în C
- operatorii logici &&, || și ! sunt disponibili în unele implementări dar nu sunt specificați de standardul POSIX
- reprezintă un limbaj de sine stătător cu blocuri de control (ex. while)
- implicit porneste un shell specializat și așteaptă comenzi

Apel

- `bc [-l] [-e expr] [file]`

# bc(1) – funcții

- $s(x)$  – sinus
- $c(x)$  – cosinus
- $e(x)$  – exponent
- $l(x)$  – logaritm
- $a(x)$  – arctangent
- $\text{sqrt}(x)$  – radical
- $\text{scale}=n$  – precizie,  $n$  numere zecimale
- $\text{quit}$  – terminare program

# Exemplu: bc(1)

Operatii de bază:

```
$ echo "1/3" | bc
```

```
0
```

```
$ echo "1/3" | bc -l
```

```
.333333333333333333333333
```

```
$ echo "1>3" | bc -l
```

```
0
```

```
$ echo "1<3" | bc -l
```

```
1
```

```
$ bc -l -e "sqrt(2)" -e quit
```

```
1.41421356237309504880
```



# Exemplu: bc(1)

```
$ bc -l
scale=4
sqrt (3)
1.7320
4*a(1)
3.1412
c(4*a(1))
- 1.0000
c(4*s(1))
-.9750
quit
```

# Translate – tr(1)

tr(1) traduce caractere primite din stdin si afisează rezultatul la stdout

string1, string2 – caracterele din primul sir sunt traduse ca cele din al doilea string

-C, -c – aplică complementul setului de caractere din string1

-d – sterge caracterele ce apar în string1

-s – elimină duplicatele traduse conform ultimul operand (fie string1 fie string2)

Tipuri de apel:

- tr [-Ccs] string1 string2
- tr [-Cc] -d string1
- tr [-Cc] -s string1

# Exemplu: tr(1)

Crează o listă de cuvinte primite la intrare:

```
$ echo "Ana are mere" | tr -cs "[A-Za-z]" "\n"
Ana
are
mere
```

Scrie textul primit cu majuscule

```
$ echo "Ana are mere" | tr "[a-z]" "[A-Z]"
ANA ARE MERE
```

Obține sirurile de caractere dintr-un fișier binar:

```
$ echo "int main(){return 0;}" | cc -xc -o test -
$ cat test | tr -cd "[:print:]"
ELF > @@8@@@@@ >>>>>>>HH H XX X @@TTTPtd44eHH H
...
```

# Stream EDitor – sed(1)

sed(1) este un editor de text în toată regula, mai mult este chiar un limbaj de programare!

- folosit în general pentru substitutii și eliminări de text
- diferit de tr(1) folosește expresii regulate
- mod de functionare
  - parcurge linie cu linie intrarea căutând un tipar
  - dacă a găsit un sir de caractere care respectă tiparul aplică funcția dată de utilizator asupra acestuia
  - funcțiile pot fi predefinite (substitutie, eliminare) sau scrise de utilizator

Apel:

- `sed [options] command [file ...]`

# Example: sed(1)

Substitutie s,tipar,text,

```
$ echo "Paul Alex Ana" | sed s,Alex,Paul,
Paul Paul Ana
```

```
$ echo "Paul Alex Alex Ana" | sed s,Alex,Paul,
Paul Paul Alex Ana
```

Substitutie s,tipar,text,flag

```
$ echo "Paul Alex Alex Ana" | sed s,Alex,Paul,g
Paul Paul Paul Ana
```

```
$ echo "Paul Alex Alex Ana" | sed s,Alex,Paul,2
Paul Alex Paul Ana
```

```
$ echo "Paul Alex Ana" | sed s,Alex,Paul,wfile
Paul Paul Ana
```

```
$ cat file
Paul Paul Ana
```

# Exemplu: sed(1)

Eliminarea linilor care contin tiparul:

```
$ echo "Paul Alex\nAlex Ana" | sed /Paul/d
Alex Ana
```

Dacă vrem să stergem doar sirul de caractere:

```
$ echo "Paul Alex\nAlex Ana" | sed s/Paul//g
Alex
Alex Ana
```

Aplicare doar anumitor linii:

```
$ echo "Paul\nAlex\nAlex\nAna\nPaul\nGeorge" | \\\nsed 1,3 s/Paul//g
```

```
Alex
Alex
Ana
Paul
George
```

# Exemplu: sed(1)

Inversarea a două cuvinte folosind expresii regulate

```
$ echo "Paul Irofti\nAlex Pop\nGeorge Stan" | sed 's
 ,^\([A-Z][A-Za-z]*\) \([A-Z][A-Za-z]*\) ,\2 \1,'
Irofti Paul
Pop Alex
Stan George
```

- `^` – caută un tipar la începutul liniei
- `[A-Z]` – trebuie să înceapă cu majusculă
- `[A-Za-z]*` – poate continua cu oricâte litere mici sau mari
- `()` – demarcă două expresii în tipar; folosim `\` ca să nu fie interpretate de shell (*escaping*)
- `\1 \2` – sirurile care au fost găsite cu cele două expresii

Există și jocuri scrise în sed(1)



<http://aurelio.net/projects/sedarkanoid/>



# awk(1)

Limbaj de programare specializat pentru procesarea datelor de tip text

- Alfred **A**ho, Peter **W**einberger și Brain **K**ernighan (1970)
- funcționează pe principiul identificare tipar - aplicare funcție ca `sed(1)`

- verifică mai mult de un tipar

```
tipar1 { comenzi }
tipar2 { comenzi }
```

- verifică linia curentă cu fiecare tipar dat după care trece la următoarea linie
- Turing Complete

Apel:

- `awk [options] [program] file ...`

# Example: awk(1)

Afisează linile care contin tiparul

```
$ echo "Paul Irofti\nAlex Pop\nGeorge Stan" | awk '/
 Pop/ {print}'
Alex Pop
```

Găseste de câte ori apare un sir în fisier:

```
$ echo "Dori si Nemo\nNemo la dentist\nDori s-a
 pierdut" | \
> awk 'BEGIN {print "Finding Nemo and Dori"}
> /[Nn]emo/ {nemo++}
> /[Dd]ori/ {dori++}
> END {print "Found Nemo " nemo " times and Dori "
 dori " times!"}'
Finding Nemo and Dori
Found Nemo 2 times and Dori 2 times!
```

Comenzile marcate cu BEGIN si END se execută doar o dată.

# Example: awk(1)

Procesarea câmpurilor dintr-un fisier

- ▶ \$0 – se referă la linia întreagă
- ▶ \$1,\$2,...,\$(10) – se referă la fiecare câmp în parte
- ▶ FS(*field separator*) – este separatorul; implicit setat ca spațiu

```
$ cat cont.txt
```

```
OP SUM
```

```
IN 10
```

```
OUT 5
```

```
IN 20
```

```
IN 3
```

```
OUT 25
```

```
$ awk '
```

```
> BEGIN {print "Fonduri disponibile"; fonduri = 0}
```

```
> /IN/ { fonduri += $2 }
```

```
> /OUT/ {fonduri -= $2 }
```

```
> END {print fonduri "RON"}' cont.txt
```

```
Fonduri disponibile
```

```
3RON
```

# awk(1): variabile si functii

## Variabile utile

- NF – numărul de câmpuri în linia curentă
- NR – numărul de linii citite până acum
- FILENAME – numele fisierului de intrare

## Functii utile

- toupper(), tolower() – litere mari, litere mici
- exp(), log(), sin() – functii matematice
- length() – lungimea sirului
- int() – partea întreagă

# Example: awk(1)

Afişează toate cuvintele:

```
$ echo "Ana are mere" | \
 awk '{ for (i=1;i<=NF;i++) print $i }'
```

Ana  
are  
mere

Listă cu utilizatori si shell folosit:

```
$ awk ' BEGIN { FS = ":" }
> {print "User " $1 " uses " $7 " shell."}'
> /etc/passwd
```

```
User nobody uses /sbin/nologin shell.
User paul uses /bin/ksh shell.
User souser uses /bin/ksh shell.
User _rsync uses /sbin/nologin shell.
User alex uses /usr/local/bin/bash shell.
```

# Example: awk(1)

Afisează doar numele shellului pentru un utilizator anume:

```
$ awk 'BEGIN { FS = ":" }
> /paul/ { cmd = "basename " $7;
> cmd | getline shell;
> print "User " $1 " uses " shell " shell."
> close(cmd);
> }' /etc/passwd
User paul uses ksh shell.
```

Execută o comandă externă si obtine rezultatul cu getline.

# Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

# Curs 8 - outline

- perspectiva utilizator (recapitulare)
- perspectiva de sistem
- servicii



# Perspectiva utilizator

- accesul utilizatorului in sistem
  - sub controlul *init*
  - *getty* aloca un terminal unui user
  - *login* autentifica utilizatorul si lanseaza un interpretor de comenzi conectat la terminalul alocat anterior
- interpretorul de comenzi
  - executa comenzi sau scripturi
  - comanda: program executabil, in executie este un proces copil ale interpretorului
  - procesele sunt imaginea din memorie a programelor de pe disc (fisiere executabile)
  - comenzile apeleaza serviciile sistemului de operare pentru:
    - gestiunea fisierelor si directoarelor
    - controlul proceselor
    - administrarea sistemului
    - *send*

# Perspectiva de sistem

- serviciile sistemului de operare accesibile in doua moduri
  - direct, prin intermediul kernelului (interfata de apeluri sistem)
  - indirect, prin intermediul unor programe specializate implementate in userspace
    - in cele din urma apeleaza tot la kernel
- comenzile shell
  - apeleaza direct serviciile kernelului, *sau*
  - contacteaza *servere* din userspace pt efectuarea serviciului
- servere
  - programe specializate care furnizeaza servicii ale sistemului de operare in spatiul utilizator
  - s.n. si *demoni* (terminologie Unix)
  - accesibile in diferite moduri printr-o forma sau alta de IPC (Inter-Process Communication)
    - memorie partajata (shared memory)
    - schimb de mesaje (message passing)
  - ex: servicii de retea, firewall, imprimare, gestiune a timpului, securitate, etc

# Servicii

- programe de sistem pornite la bootarea sistemului de operare
  - fie servere/demoni, fie programe care contribuie la buna functionare a sistemului si la asigurarea mediului de executie pentru programele utilizator
  - tehnic, procese pornite de *init*
  - furnizeaza diverse servicii de sistem utilizatorului
    - direct, prin servere/demoni
    - indirect, prin asigurarea unui mediu de executie corespunzator programelor utilizator
  - in general, ruleaza atata vreme cat sistemul e in functiune (eventual restartate automat, daca apar erori)
  - configurate pt fiecare runlevel in parte
  - operatii standard: pornire, oprire, reincacarea configuratiilor, afisarea starii
- multiple interfete de acces si gestiune
  - *System V* (Unix)
  - *Upstart* (implementare Linux pt *init*)
  - *Systemd* (varianta recenta Linux)

# Interfata System V

- configurata in fisierul */etc/inittab* in sistemele Unix (si versiuni mai vechi de Linux)
  - *init* citeste aici runlevel-ul default (*initdefault*)
- servicii grupate in directoare de tip *rc* (*run commands*), cate unul pentru fiecare runlevel + unul pt etapa initiala, imediat dupa boot

*/etc/rc0.d, /etc/rc1.d, /etc/rc2.d, ..., /etc/rc6.d, /etc/rcS.d*

(in Unix grupate in */etc/rc.d/*)

- fiecare director *rc* contine link-uri simbolice catre scripturi din */etc/init.d* care contin serviciile executate la intrarea sistemului in runlevel-ul respectiv
  - ordinea executiei data de ordinea lexicografica a numelor de servicii
- dezactivare serviciu intr-un anumit runlevel
  - redenumirea link-ului catre script a.i. noul nume incepe cu K+2 cifre  
`$ ln -s ../init.d/ssh K01ssh`
- activare serviciu intr-un anumit runlevel
  - redenumirea link-ului catre script a.i. noul nume incepe cu S+2 cifre  
`$ ln -s ../init.d/rc.local S05rc.local`

# Scripturi init.d

- includ la inceput, dupa specificarea interpretorului, un header cu informatii de configurare (in Linux, definite cf. LSB, Linux Standard Base)

- tehnic sunt comentarii, dar interpretate la init
- cuprinse intre liniile urmatoare

*### BEGIN INIT INFO*

*### END INIT INFO*

- fiecare linie de tipul

*# {keyword}: arg1 [arg2 ...]*

- keyword-uri de pornire/oprire serviciu

- definesc runlevel-urile in care serviciile trebuie pornite/oprite implicit

*# Default-Start      2 3 4 5*

*# Default-Stop      0 1 6*

# Comenzi manipulare interfata SysV

- automatizarea lucrului cu directoarele *rc*
  - diverse comenzi in functie de SO: *chkconfig*, *update-rc.d/bum*, etc

Ex: Ubuntu, dryrun

```
$ update-rc.d -n ssh enable 2 3 4 5
```

- executia operatiilor serviciilor din */etc/init.d*

```
$ service --status-all
```

```
$ service ssh start
```

```
$ service ssh stop
```

```
$ service ssh reload
```

```
$ service ssh restart
```

# Upstart

- implementare *init* in Linux
  - event-driven
  - evenimentele sunt mesaje trimise catre servicii
- procesele de serviciu s.n. *job*-uri si sunt configurate in */etc/init*
  - *job*-urile sunt actiuni executate ca urmare primirii unor mesaje (evenimente)
  - serviciile configurate au precedenta fata de serviciile din */etc/init.d*
  - modificarea fisierelor de configurare monitorizata cu *inotify*
  - fisierele de configurare includ informatii despre pornirea/oprirea joburilor

Ex:                   *start on runlevel [2345]*  
                      *stop on runlevel [!2345]*

- alte informatii de configurare: *respawn* (job repornit automat in caz de terminare anormala), *exec*

Ex, *tty1.conf*:

```
...
respawn
exec /sbin/getty -8 38400 tty1
```

# Upstart (cont.)

- evenimente standard:
    - *startup* (compatibilitate System V, porneste joburi cu evenimente tip *runlevel*) emis la pornirea *init*
    - la pornirea/oprirea job-urilor, *init* emite *starting*, *started*, *stopping*, *stopped*
  - functionare
    - (1) incarca din */etc/init* configuratia job-urilor
    - (2) pt eveniment de start, ruleaza jobul corespunzator
    - (3) joburile create genereaza noi evenimente care la randul lor pornesc alte joburi
    - (4) continua acest ciclu pana cand se termina toate joburile necesare
  - spre deosebire de System V, permite si gestiunea unor sesiuni utilizator
    - *user session mode*
    - $PID > 1$
    - fisierele de configurare servicii: *\$XDG\_CONFIG\_HOME/upstart*, *\$HOME/.init/*  
*\$XDG\_CONFIG\_DIRS/upstart*, */usr/share/upstart/sessions*
- Ex:   \$ upstart -user*



# Initctl

- comanda de lucru cu *upstart*
- vizualizare joburi

*\$ initctl list*

# lista de joburi si starea asociata

*\$ initctl status networking*

# vizualizare job specific

- stare job: pereche *scop/stare curenta*
  - scopul defineste operatia dorita (start, stop, etc)
  - perechea se actualizeaza pe masura ce joburile evolueaza
- pornire/oprire joburi

*\$ initctl start networking*

*\$ initctl stop networking*

- emitere manuala eveniment (mesaj)

*\$ initctl emit some\_event*

# Systemd

- versiunea recenta de *init* pt Linux
- foloseste tinte (*targets*) in loc de runlevels pentru a porni serviciile sistem
  - target-urile au dependente care trebuie indeplinite
  - sistemul de dependente gestioneaza 12 tipuri de unitati (*units*)
- unit
  - incapsuleaza diferite obiecte relevante pentru bootare si mentenanta sistemului
  - descrise in fisiere de configurare
  - pot fi *active/inactive*, similar cu procesele (au si stari intermediare)
- ex tipuri de unitati
  - *service units* – pornesc si controleaza demonii
  - *target units* – grupeaza unit-uri sau furnizeaza puncte de sincronizare la bootare
  - *device units* – expun echipamentele in *systemd* si activarea bazata pe echipamente
  - *mount units* – controleaza mountpoint-urile din sistemul de fisiere
  - *timer units* – utile pt a activa alte unit-uri pe baza timerelor
  - *samd*.

# Systemd (cont.)

- serviciile nu sunt pornite intr-o ordine anume ca in System V, ci pe baza dependentelor unitatilor
- la fel ca si *upstart*, poate fi pornit in mod utilizator (PID > 1)

Ex:      *\$ systemd -user*

- compatibil cu System V in mare masura
  - scripturile *init.d* sunt suportate si folosite ca alternativa de configurare
  - furnizeaza interfata */dev/initctl* sau */run/initctl* (fisier FIFO pt comenzi *upstart*)
  - suport pentru tool-urile System V
- functionare *systemd*
  - incarca fisierele de configurare a serviciilor din */etc/system/system* sau */lib/systemd/system* (si/sau */usr/lib/systemd/system*)
  - determina tinta de boot (*boot target*), uzual *default.target*
  - determina dependentele boot target si le activeaza

# Systemd targets

- echivalenta runlevels
  - *poweroff.target* – shutdown
  - *rescue.target* – single user mode
  - *multi-user target* – multi-user + networking
  - *graphical.target* – multi-user + networking + GUI
  - *reboot.target* – reboot
  - *default.target* – tinta implicita de boot, link simbolic cate una din tintele de mai sus
- ex: bootare in default-target
  - activeaza unit-urile dependente: *networking.service*, *crond.service*, etc
- Obs: tintele de bootare nu sunt singurele controlate de *systemd*

# Systemd units

- fisiere de configurare unitati
  - codifica informatii despre servicii, echipamente, mountpoint-uri, etc
  - inspirate de XDG Desktop Entry Specification (gen fisiere *.desktop*)
  - contin mai multe sectiuni: [Unit], [Install] + sectiuni specifice (e.g. [Service] pt service units)
- [Unit]
  - descriere unit
  - controleaza ordinea si/sau timpul activarii
- [Service]
  - aici se pornesc, opresc, reincarca serviciile
- [Install]
  - folosita pentru exprimarea dependentelor

# Exemplu fisier configurare unit ssh.service

## [Unit]

Description=OpenBSD Secure Shell server

After=network.target auditd.service

ConditionPathExists=!/etc/ssh/sshd\_not\_to\_be\_run

## [Service]

EnvironmentFile=-/etc/default/ssh

ExecStartPre=/usr/sbin/sshd -t

ExecStart=/usr/sbin/sshd -D \$SSHD\_OPTS

ExecReload=/usr/sbin/sshd -t

ExecReload=/bin/kill -HUP \$MAINPID

KillMode=process

Restart=on-failure

RestartPreventExitStatus=255

Type=notify

## [Install]

WantedBy=multi-user.target

Alias=sshd.service

# Comenzi systemd

- implementate de *systemctl*

- listare unitati

*\$ systemctl list-units*

- pornire/oprire/repornire unitati

*\$ systemctl start ssh.service*

*\$ systemctl stop ssh.service*

*\$ systemctl restart ssh.service*

- activare/dezactivare unitati

*\$ systemctl enable ssh.service*

*\$ systemctl disable ssh.service*

# Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru



# Curs 9 - outline

- demoni
- gestiunea timpului
  - cron
  - anacron
  - at

# Demoni

- procese (servere) care ruleaza in background si nu sunt asociate cu un terminal de control
- in general porniti ca servicii de sistem la sfarsitul operatiei de boot
- serverele pornite de utilizator dintr-un shell (terminal) se dezasociaza de terminal pt a evita
  - interactiuni nedorite cu controlul proceselor executat de shell
  - interactiunea cu gestiunea sesiunii de terminal
  - tiparirea mesajelor pe terminal (avand in vedere ca ruleaza in background)
- exemple
  - serviciile sistem din */etc/rc.d* pornite cu privilegii de superuser (*inetd/xinetd*, *Web*, *sendmail/postfix*, *syslogd* etc)
  - job-uri *cron/at* invocate de demonul de *cron/at*
  - servere/programe pornite din terminal

# Transformarea programelor in demoni

- (1) detasarea de terminal
- (2) schimbarea directorului de lucru curent
  - demonul schimba directorul de lucru curent in / (sau alt director la alegere)
  - fisierele core generate de demon se salveaza in directorul de lucru curent
  - daca directorul de lucru al demonului era pe un sistem de fisiere si demonul continua sa lucreze acolo, sistemul de fisiere respectiv nu va putea fi dezinstalat (*umount*)
- (3) inchiderea tuturor descriptorilor de fisiere deschisi
  - se inchid toti descriptorii de fisiere mosteniti de la procesul care a creat demonul (in mod normal, shell-ul)
- (4) redirectarea *stdin*, *stdout* si *stderr* la */dev/null*
  - garanteaza ca descriptorii de fisiere corespunzatori sunt deschisi, dar citirea intoarce EOF iar scrierea se pierde
  - necesar pt ca functiile de biblioteca care presupun folosirea acestor descriptori sa nu esueze
- (5) utilizarea *syslogd* pentru logarea erorilor

# Syslogd

- fara terminal de control, demonii necesita metode specifice pt a afisa mesaje (ex mesaje de eroare, urgenta, logging, etc)
- metoda standard foloseste functia *syslog* care trimite mesaje catre demonul *syslogd* (*rsyslogd* pe sisteme Linux)
  - fisier de configurare *syslog.conf*, specifica ce trebuie facut cu fiecare mesaj primit de demon:
    - se adauga la sfarsitul unui fisier
    - se logheaza pe consola (*/dev/console*)
    - se forwardeaza catre demonul *syslogd* al altei masini
  - serviciul functioneaza pe portul 514 UDP (v. */etc/services*)
  - la primirea semnalului SIGHUP se reciteste fisierul de configurare (in sistemele moderne se foloseste comanda *service reload*)
    - ex: `$ kill -HUP <pid syslogd>`
  - controlat de *init/systemd* prin interfetele discutate anterior
- mesajele se pot trimite direct pe portul 514 UDP, dar uzual se foloseste functia *syslog* sau comanda *logger*

# Functia syslog

- in lipsa terminalului de control demonul nu poate folosi functii de genul *fprintf* la *stderr*
- functia *syslog*, respectiv comanda *logger* permit scrierea mesajelor in *log-uri*
  - uzual, fisiere din */var/log* sau *consola*
- mesajele logate sunt caracterizate de o combinatie intre *nivel* si *facilitate* (*level/facility*)
  - combinatia se mai numeste si *prioritate*
- nivelul mesajelor
  - valoare intre 0 si 7
  - determina importanta mesajelor: urgenta, alerta, critice, eroare, atentionare, etc
  - permite ca toate mesajele de un anumit nivel sa fie tratate unitar
  - valoare implicita: LOG\_NOTICE (nivel 5, notificare normala dar semnificativa)

# Functia syslog (cont.)

- *facilitate*
  - specifica tipul de program care trimite mesajul
  - ex: mesaje kernel, de autentificare, temporale (*cron/at*), mail, imprimanta, etc
  - valoarea implicita: LOG\_USER (mesaje generice de nivel utilizator)
  - valoarea facilitatii permite ca toate mesajele venite de la o anumita facilitate sa fie tratate la fel in *syslog.conf*
- ex:

|              |                    |
|--------------|--------------------|
| kern.*       | /dev/console       |
| local7.debug | /var/log/cisco.log |

# Utilizare syslog

- vizualizarea mesajelor logate in timp real

*\$ tail -f /var/log/syslog*

- *head/tail*
  - afiseaza partea de inceput, respectiv sfarsit a unui fisier
  - uzual afisarea se face in termen de linii, cu optiunea *-n*

*\$ head -n 5 /etc/passwd*

- logarea mesajelor din linie de comanda

*\$ logger -p user.info Some user message of certain importance*

# Cron

- demon care executa comenzi planificate pentru rulare la un moment dat (uzual periodic)
- pornit ca serviciu de sistem la bootare
  - varianta interactiva, nedemonizata  
*\$ cron -f*
- incarca in memorie tabele cu planificarile taskurilor utilizator, *crontabs*
  - disponibile in */var/spool/cron/crontabs/\$USER*
  - create/editate cu comanda *crontab*
  - */etc/crontab*, tabela de sistem (cu variantele */etc/cron.hourly*, */etc/cron.daily*, */etc/cron.weekly*, */etc/cron.monthly*)



# Functionare crond

- la fiecare 60s examineaza continutul crontab-urilor pt a stabili daca respectivele comenzi trebuie rulate in timpul minutului current
- rezultatele executiei comenzilor sunt trimise pe mail proprietarului crontab sau utilizatorului specificat in MAILTO
- in plus, in fiecare minut verifica daca directorul de spool sau tabela de sistem s-au modificat
  - daca da, reincarca continutul tuturor crontab-urilor
  - *crontab* modifica implicit si timpul de modificare al directorului de spool

# Crontab

- comanda pentru gestiunea tabelelor utilizator
- in general, tabelele nu sunt editate direct
- instalarea unei tabele crontab

*\$ crontab <file>* # fisierul contine planificarea in format specific

- */etc/cron.allow* si */etc/cron.deny* se pot folosi pt controlul permisiunii de a folosi un crontab
  - format: un nume de utilizator singur pe o linie
  - daca nu exista, depinde de setarile sistemului, fie doar root-ul poate folosi crond, fie toti utilizatorii
  - precedenta: allow, deny

# Operatii crontab

- editare
  - lanseaza \$VISUAL sau \$EDITOR sau */usr/bin/editor*

*\$ crontab -e*

- stergere

*\$ crontab -r*

- afisare

*\$ crontab -l*

# Formatul crontab

- spatiul, newline ignorate
- # marcheaza comentarii
- linii active de doua feluri
  - setari de variabile de mediu
  - comenzi
- dereferentierea variabilelor de mediu nu functioneaza

Ex:      `$PATH=$HOME/bin:$PATH`

- variabile de mediu recunoscute:
  - SHELL=/bin/sh implicit
  - LOGNAME, HOME preluate din /etc/passwd (LOGNAME nu se poate schimba)
  - PATH=/usr/bin:/bin , valoare implicita
  - MAILTO, valori: utilizator, lista de utilizatori, nedefinita implica \$LOGNAME

# Formatul crontab (cont.)

- `<min> <ora> <zi> <luna> <zi a sapt.> <cmd> “\n”`
- campurile referitoare la timp pot fi:
  - valori uzuale, ex: min: 0-59, ore: 0-23, zi: 1-31, luna:1-12, zi a sapt.: 0 – 7, mon-sun (0 si 7 duminica, mon, tue, wed, samd.)
  - intervale, ex: 8-11 pt ore inseamna orele 8,9,10,11
  - liste, ex: “1,2,5,9”, “0-4,8-12” pt zilele lunii
  - pasi (steps), ex: “0-23/2” pt ore, din 2 in 2 ore
  - “\*”: wildcard, semnificatia fiind intreg intervalul posibil (ex: 0-23 pt ore, 0-31 pt zile)
- comenzi
  - executate de /bin/sh sau \$SHELL
  - “%” : prima aparitie e newline, dupa el stdin
  - “%” poate fi escaped cu “\”

# Exemplu crontab

```
utilizeaza /bin/bash pt a rula comenzi, in loc de /bin/sh
SHELL=/bin/bash
utilizatorul catre care se trimit pe mail rezultatele
MAILTO=paul
#
ruleaza la 5 minute dupa miezul noptii in fiecare zi
5 0 * * * $HOME/bin/daily.job >> $HOME/tmp/out 2>&1
ruleaza la 2:15pm in prima zi a lunii
15 14 1 * * $HOME/bin/monthly
ruleaza la 10 pm in zilele de lucru deranjandu-l pe Joe
0 22 * * 1-5 mail -s "It's 10pm" joe%Joe,%%Where are your kids?%
23 0-23/2 * * * echo "run 23 minutes after midn, 2am, 4am ..., everyday"
5 4 * * sun echo "run at 5 after 4 every sunday"
ruleaza fiecare a doua sambata din luna
0 4 8-14 * * test $(date +%u) -eq 6 && echo "2nd Saturday"
```

# Anacron

- cron necesita functionarea in permanenta a calculatorului
- *anacron*
  - ruleaza comenzi periodice, cu frecventa precizata in zile
  - nu pp. rulara continua a sistemului
  - lista de job-uri in */etc/anacrontab*
  - format linie: *<perioada in zile> <delay in min> <job id> <comanda>*
  - verifica daca s-a executat comanda in ultima perioada (nr de zile de mai sus)
    - daca nu, se executa comanda dupa delay-ul specificat
  - la terminarea comenzii, se salveaza data (fara ora) intr-un fisier in */var/spool/anacron* pt job-ul respectiv pt a se sti cand sa se execute din nou
  - job id-ul trebuie sa se potriveasca cu unul dintre argumentele comenzii
  - rezultatul job-urilor e trimis fie catre *root* fie catre *\$MAILTO*, daca exista

# At/atd

- comanda folosita pentru controlul executiei intarziate a comenzilor
- comenzile sunt plasate in cozi de asteptare (desemnate prin litere: a-z, A-Z)
- *at* – executa o comanda la un timp specificat
- *atq* – listeaza job-urile utilizatorului programate pt executie ulterioara aflate intr-o anumita coada (*a* e coada implicita pt *at*)
- *atrm* – sterge un job programat pt executie ulterioara
- specificarea timpului
  - HH:MM, MM/DD/YY, DD.MM.YY, *samd*
  - *now* + nr unitati

Ex: “4pm + 5 days”, “11am Jun 23”, “3pm tomorrow”, etc



# Exemplu at

```
$ at now + 1 minute
```

```
warning: commands will be executed using /bin/sh
```

```
at> ps auxw
```

```
at> <EOT>
```

```
job 4 at Mon Nov 25 20:07:00 2024
```

```
$ atq
```

```
4 Mon Nov 25 20:07:00 2024 a guest
```

```
$ atrm 4
```

```
$ atq
```

```
$
```

# Gestiunea timpului, ceasul HW

- doua ceasuri: HW si sistem
- *hwclock*
  - comanda pt accesul la ceasul HW al sistemului (RTC, CMOS)
  - afiseaza/seteaza timpul HW
    - \$ hwclock --set --date*
  - seteaza ceasul HW la valoarea ceasului sistem si invers
    - \$ hwclock --hctosys*
    - \$ hwclock --systohc*
  - compenseaza drift-ul ceasului HW (folosind istoria din */etc/adjtime*)
    - \$ hwclock --adjust*
  - compara cele doua ceasuri
    - \$ hwclock -c*      # compara periodic (10s)
  - prezice valori viitoare ale ceasului HW pe baza vitezei de drift
    - \$ hwclock --predict*
- *samd*

# Ceasul sistem

- componenta a kernelului bazata pe un timer (intrerupere de timp)
- are sens doar cat merge calculatorul
- reprezinta nr de secunde de la 1 Ian 1970 00:00:00
- ceasul care conteaza in sistem, initializat la boot din ceasul HW
- *date*
  - afiseaza/seteaza ceasul sistem
  - foloseste parametri de formatare introdusi cu “+”

|                    |                                  |
|--------------------|----------------------------------|
| <i>\$ date +%d</i> | # afiseaza ziua lunii (e.g., 02) |
| <i>\$ date +%m</i> | # luna                           |
| <i>\$ date +%y</i> | # anul                           |
| <i>\$ date +%H</i> | # ora                            |
| <i>\$ date +%M</i> | # minutul                        |

samd.

# Timpul de executie al unei comenzi

- masurarea timpului de executie al unei comenzi

*\$ time find / -name crontab*

*\$ time sleep 5*

- *time* e comanda interna shell, dar exista si varianta externa
  - afiseaza trei timpi:
    - timpul real
    - timpul petrecut in spatiul utilizator
    - timpul petrecut in kernel
  - in plus, sumarizeaza folosirea resurselor sistemului
    - se foloseste un format specific
    - in *bash* se foloseste comanda externa */usr/bin/time*

# Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

# Curs 10 - outline

- stocarea datelor
  - echipamente de tip bloc
  - sisteme de fisiere

# Stocarea datelor

- memoria principala – singurul mediu de stocare de dimensiune mare accesibil direct procesorului
  - acces random
  - uzual volatila
  - tipic Dynamic Random-Access Memory (DRAM)
- stocarea secundara – extensie a memoriei principale care furnizeaza capacitate mare de stocare **nevolatila**
  - discuri dure (hard disks)
  - discuri flexibile (floppy disks)
  - CDROM/DVD
  - SSD (Solid State Disks)
  - flash drives
  - samd

# Stocarea datelor (cont.)

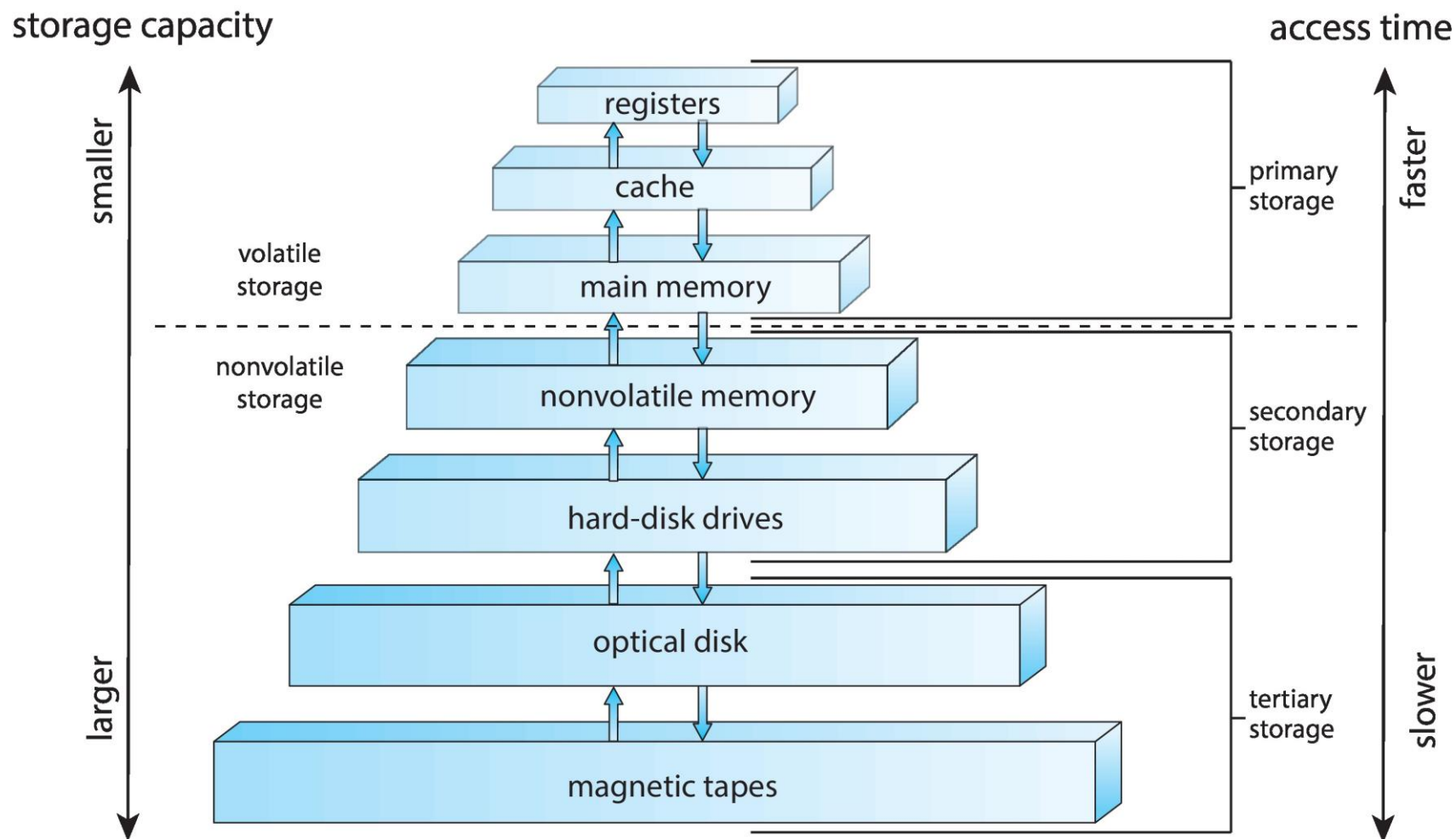
- **discuri dure (Hard Disk Drives, HDD)** – platane rigide din metal sau sticla acoperite cu material magnetic capabil de stocarea datelor
  - platanele se invart in jurului unui ax de rotatie
  - suprafata unui platan divizata logic in **piste**
  - pista divizata la randul ei in **sectoare**
  - uzual exista mai multe platane => in 3D colectia de piste egal departate de axul de rotatie formeaza un **cilindru**
  - controller-ul de disc (**disk controller**) determina interactiunea logica dintre echipament si calculator
- **echipamente cu memorie nevolatila (Non-volatile memory, NVM)**– mai rapide decat HDD, nevolatile
  - diverse tehnologii
  - devin din ce in ce mai raspandite



# Ierarhia de memorie

- sistemele de stocare organizate in ierarhii in functie de
  - viteza
  - cost
  - volatilitate
- **caching**
  - copiaza informatia in sisteme de stocare mai rapide
  - memoria principala vazuta drept cache pentru stocarea secundara
- **driverul de echipament (Device Driver)**
  - componenta kernel specializata, opereaza fiecare device controller pt a gestiona operatiile de I/O
  - furnizeaza o interfata uniforma intre controller si kernel

# Ierarhia echipamentelor de memorie



# Echipamente de tip bloc in Unix

- unitatea de transfer a datelor = blocul de date
- arhitecturile de calcul moderne bazate pe Direct Memory Access (DMA)
  - transfer de date intre RAM si echipamentul bloc neintermediat de CPU
  - suport HW sub forma unui chip programabil specializat
  - contrar tehnicii de *programmed I/O*
- reprezentate in sistemele Unix ca fisiere speciale de tip bloc in directorul */dev*

Ex:        */dev/sda1, /dev/cdrom*

- caracterizate de *nr major* si *nr minor*
  - primul identifica driverul asociat echipamentului, al doilea nr unitatii de acel tip din sistem
- create cu *mknod*

Ex:        *\$ mknod /dev/sdc1 b 8 1*

# Echipamente loop (loop devices)

- *pseudo-echipament (pseudo-device)* care permite folosirea unui fisier obisnuit (*regular file*) ca echipament bloc
  - ex: sistem de fisiere continut intr-un singur fisier
    - imagini ISO de CDRom/DVD sau imagini floppy disk
    - se pot instala in sistem cu comanda *mount* fara suport HW
  - uzual */dev/loop0*, */dev/loop1*, samd.
  - *losetup*
    - asociaza device-uri loop cu fisiere obisnuite sau echipamente bloc
- Ex:
- |                                     |                                          |
|-------------------------------------|------------------------------------------|
| <code>\$ losetup -f file.img</code> | # asociaza primul loop device disponibil |
|                                     | # cu fisierul furnizat ca argument       |
| <code>\$ losetup -j file.img</code> | # afiseaza starea loop devices asociate  |
|                                     | # cu fisierul imagine argument           |

# Utilizarea echipamentelor bloc

- stocarea datelor
  - datele inregistrate pe discuri cf unui format specific unui anumit tip de sistem de fisiere
  - uzual, formatul genereaza structura logica de acces cu fisiere si respectiv colectii de fisiere (directoare)
  - inregistrarea formatului sistemului de fisiere pe disc (“formatarea discului”)
    - se folosesc comenzi speciale pt fiecare tip de sistem de fisiere

Ex:            `$ mkfs -t ext4 /dev/sda1`        # ⇔ `$ mkfs.ext4 /dev/sda1`
- spatiu de swap
  - discuri neformatate (in sensul de mai sus) folosite de memoria virtuala a sistemului de operare
  - programele utilizator pot fi mai mari decat memoria RAM disponibila
  - memorie RAM insuficienta pt noi procese
  - creat cu *mkswap*, activat cu *swapon*, dezactivat cu *swapoff*

Ex:            `$ mkswap /dev/sda2 ; swapon /dev/sda2`

# Comenzi utile

- *parted/fdisk* - manipuleaza tabela de partitii a unui disc

*\$ parted -l /dev/sda*      # afiseaza tabela de partitii a /dev/sda

- *blkid/lsblk* – furnizeaza informatii/attribute ale echipamentelor bloc

*\$ blkid; lsblk -f*      # attribute discuri + informatii despre FS

- *df* – raporteaza informatii despre utilizarea spatiului de disc

*\$ df -h*      # afisare “human readable”

- *du* – estimeaza utilizarea spatiului folosit de fisiere

*\$ du -sh \**      # dim totala a fisierelor din directorul curent

- *free* – afiseaza memoria si spatiul de swap disponibile

*\$ free -h*      # human readable

# Fisiere

- fisier
  - abstractie de nivel de sistem de operare pt stocarea persistenta a datelor
  - concret, containere pt stocarea persistenta a datelor
  - la nivelul cel mai de jos, stocarea persistenta se face pe discuri
  - uzual referite prin nume (string ASCII) convertit la o reprezentare interna a kernelului de catre sistemul de fisiere
  - paradigma uzuala de folosire: *open – read/write – close*
- pe langa date, fisierele stocheaza si *metadata*:
  - attribute: data ultimului acces, ultimei modificari, proprietarul fisierului, permisiuni, dimensiunea fisierului, etc
  - structura de acces la reprezentarea low-level a datelor (adrese de blocuri de disc)

# Sistemul de fisiere

- componenta speciala a SO care gestioneaza fisierele si directoarele
- gestioneaza mediul de stocare persistenta a datelor
  - structureaza datele pe disc intr-un anumit *format*
- ofera utilizatorului o interfata uniforma de acces la date
  - bazata pe abstractia de fisier si operatiile (apeluri sistem) de lucru cu ele
- SO moderne capabile sa integreze sisteme de fisiere cu format diferit in aceeasi ierarhie de directoare
  - VFS – Virtual Filesystem Switch (ext3, ext4, ntfs, vfat, etc)
- disponibil utilizatorului ca urmare a operatiei de *mount*

*\$ mount -t ext4 /dev/sda1 /*

*\$ umount /dev/sda1*      # operatia inversa, merge si *umount /*

- directorul in care se instaleaza discul formatat s.n. *mountpoint*



# /etc/fstab

- tabela system-wide cu mountpoint-uri inspectata la bootarea SO
- la bootare, mountpoint-urile din tabela se instaleaza ca si cand s-ar fi apelat

*\$ mount -a*

- suplimentar, *systemd* instaleaza unit-urile de tip *mount*
- mountpoint-urile active la un moment dat disponibile in */etc/mtab* sau */proc/mounts*

*\$ mount*                      # afiseaza informatia din tabele

- fiecare mountpoint si sistemul de fisiere asociat sunt descrise pe o linie din *fstab*
  - fiecare linie contine 6 campuri

# Campurile fstab

- (1) *fs\_spec* – echipamentul bloc sau sistemul de fisiere remote care trebuie instalat

Ex:        */dev/sda7, /dev/cdrom, fmi.unibuc.ro:/home*

- se pot folosi LABELS sau UUIDs
    - metoda preferabila, numele de device se poate schimba cand se adauga/indeparteaza device-uri
- \$ blkid*                      # sau *lsblk -f*

- (2) *fs\_file* – mountpoint-ul (sau *none*, pt partitia de swap)
- (3) *fs\_vfstype* – tipul de sistem de fisiere: *ext4, xfs, vfat, ntfs, nfs, proc*, etc (*swap* denota fisierul sau partitia de swap)
- (4) *fs\_mntops* – optiunile operatiei *mount* pt sistemul de fisiere respectiv
  - *defaults* inseamna *rw, suid, dev, exec, auto, nouer, async*
  - *user/owner*, permite operatia de *mount* pt utilizator/proprietar
  - *noauto*, mountpoint-ul nu e instalat inb sistem la operatia *mount -a*

# Campurile fstab (cont.)

- (5) *fs\_req* – folosit pt backup (*dump*), uzual 0
- (6) *fs\_passno* – utilizat de *fsck* pt a determina ordinea de verificare la boot
  - radacina / trebuie sa aiba valoarea 1
  - celelalte mountpoint-uri valoarea 2
  - valoare implicita 0 (nu se executa *fsck* la boot), uzual pt remote FS

Obs: *fsck* verifica (si repara daca se poate) un sistem de fisiere

Ex: `$ fsck -t ext4 /dev/sdb2`

`$ fsck /dev/sda1`

`$ fsck /home`

De asemenea, *fsck* poate folosi LABELs/UUIDs

# Tipuri de sisteme de fisiere

- sisteme de fisiere pt date stocate permanent
  - log-structured/journaling sau nu
- sisteme temporare
- sisteme de fisiere bazate pe echipamente loop
- sisteme de fisiere distribuite
- pseudo-sisteme de fisiere

# Log-structured/journaling FS

- actualizarea FS pp multe scrieri
- intreruperi nedorite (system crash, intreruperea curentului) in mijlocul acestor scrieri pot lasa structurile de date interne ale FS-ului intr-o stare inconsistenta
- detectarea acestor inconsistente si recuperarea din starea de inconsistenta pp analiza exhaustiva a structurilor de date FS
  - uzual, implica rulara *fsck*
  - se face inainte ca FS sa fie instalat si folosit din nou pt R/W
  - daca FS are dimensiuni mari si/sau I/O bandwidth-ul e redus => timpi mari de downtime pt sistem
- solutie: *log-structured/journaling FS*
  - aloca un spatiu special (*log/journal*) care stocheaza modificarile care vor fi operate *ulterior* asupra FS
  - dupa crash, se citeste log-ul si se reiau operatiile salvate in log pana cand structurile de date interne FS devin consistente din nou

# Log-structured/journaling FS (cont.)

- concret: FS inregistreaza fiecare actualizare a metadatelor ca o tranzactie
- toate tranzactiile sunt scrise intr-un *log/journal*
  - tranzactia e comisa de indata ce a fost scrisa (secvential) in log
  - uneori se foloseste un device separat sau o anumita sectiune a discului
  - in orice caz, in acest moment FS nu poate fi actualizat
- tranzactiile din log sunt scrise *asincron* in structurile de date ale FS
  - cand o asemenea structura s-a modificat, tranzactia e stearsa din log
- la *crash*, toate tranzactiile ramase in log trebuie executate din nou
- consecinte:
  - (1) recuperarea rapida din crash
  - (2) indeparteaza posibilitatea aparitiei inconsistentelor in metadate
- exemple: *ext4*, *xfs*, *reiserfs*, *samd*

# Temporary FS (tmpfs)

- sistem de fisiere volatil, stocat in RAM
  - datele stocate in *tmpfs* se pierd dupa dezinstalarea FS (*umount*), dupa reboot sau la caderea curentului electric
- nu este RAM disc, are structura logica de FS, cu abstractii de nivel inalt tip fisiere si directoare
  - RAM disc – disc virtual, organizat dupa principiile HDD
  - eventual, un FS poate rula peste RAM disc
- beneficiaza de spatiul de swap !
  - rezolva problema limitelor de alocare (*out-of-memory*)
- tipul VFS: *tmpfs*

Ex: `$ mount -t tmpfs ramfs /mnt -o size=1g`

- Obs: nu exista device in comanda, inlocuit de un string ales cf dorintei utilizatorului (i.e., *ramfs*)
- utilitate: stocarea datelor temporare (date de configurare severe, FIFO pt IPC programe de sistem, etc), sisteme de fisiere speciale gen *cgroups*, etc

# Loop device FS

- prin asocierea unui fisier cu un loop device, continutul fisierului poate fi folosit impreuna cu comanda *mount*
- Ex: CDROM mount

```
$ mount -o loop -t iso9660 cdrom.iso /cdrom
```

- Ex: spatiu de swap care foloseste un fisier in loc de device

```
$ dd if=/dev/zero of=~ /myswapfile bs=1K count=1M
```

```
$ losetup --show -f ~/myswapfile # pp device-ul ales e /dev/loop0
```

```
$ mkswap /dev/loop0
```

```
$ swapon /dev/loop0
```

```
$ swapon
```



# Loop device FS (cont.)

- Ex: crearea unui sistem de fisiere intr-un fisier

```
$ dd if=/dev/zero of=~ /ext4.img bs=1K count=1M
```

```
$ losetup --show -f ~/ext4.img # pp device-ul ales e /dev/loop0
```

```
$ mkfs.ext4 /dev/loop0 # creeaza format ext4
```

```
$ file ~/ext4.img
```

```
$ mount /dev/loop0 /mnt
```

*...*

```
$ umount /dev/loop0
```

```
$ losetup -d /dev/loop0
```

# Sisteme de fisiere paralele/distribuite

- sisteme message passing, paradigma client-server
- FS server
  - program remote care exporta (o parte a) FS local la distanta
  - uzual se exporta un director de pe masina remote
- FS client
  - program client care serveste ca intermediar intre FS server si VFS-ul local
- Ex: Network File System (NFS)
  - bazat pe Remote Procedure Calls (RPC), executa proceduri la distanta ca si cum ar fi locale
  - procedura locala (i.d., *read*) executata de un *stub client* care contacteaza FS server pt executia procedurii reale la distanta (*stub server*)
  - parametrii de apel si rezultatul operatiei transformati intre formatul de date local si cel de retea (XDR, ASN.1) si inapoi (operatii de *marshalling/unmarshalling*)

*\$ mount -t nfs fmi.unibuc.ro:/home /home*

# Pseudo-sisteme de fisiere

- in general, interfete catre structurile de date ale kernelului
- *procfs*

*\$ mount -t proc proc /proc*

- in principal, contine subdirectoare pentru fiecare proces din sistem de forma */proc/[pid]*
  - intr-un subdirector *[pid]* se gasesc informatii despre proces, cum ar fi
    - linia de comanda folosita pentru a lansa procesul
    - cuset-ul procesului
    - cwd, environment, link executabil, file descriptori deschisi (inclusiv 0,1,2)
    - statistici operatii I/O ale procesului
    - acces la paginile de memorie ale procesului + pagemap-ul procesului
    - sistemele de fisiere montate in namespace-ul procesului
    - starea procesului folosita de comanda *ps*
    - informatii despre utilizarea memoriei (dim, dim rezidenta, text, date, lib)
    - descrierea apelului sistem curent (argumente, SP, PC)
    - informatii despre thread-urile si timerele procesului
- samd.

# Pseudo-sisteme de fisiere (proc)

- in plus, */proc* contine informatii despre
  - echipamentele PCI ( */proc/bus/pci/devices*, vizibile cu comanda *lspci*)
  - linia de comanda a kernelului la lansare ( */proc/cmdline*)
  - informatii despre CPU si arhitectura lor ( */proc/cpuinfo*)
  - lista nr majore ale echipamentelor ( */proc/devices* )
  - lista sistemelor de fisiere suportate de kernel ( */proc/filesystems* )
  - incarcarea medie a sistemului afisate de comanda *uptime* ( */proc/loadavg* )
  - statistici despre utilizarea memoriei afisate de comanda *free* ( */proc/meminfo* )
  - lista modulelor incarcate in sistem ( */proc/modules* )
  - lista sistemelor de fisiere curent instalate in sistem ( */proc/mounts*, v. *mount*)
  - informatii despre reseaua sistemului, subdirectorul */proc/net*
  - lista partițiilor de disc din sistem ( */proc/partitions* )
  - statisticile kernelului ( */proc/stat* )
  - partițiile de swap in folosinta curenta ( */proc/swaps* )
  - valorile diverselor variabile ale kernelului ( */proc/sys* )
  - statistici despre sistemul de memorie virtuala ( */proc/vmstat* ), samd.

# Pseudo-sisteme de fisiere (sysfs)

- *sysfs*

*\$ mount -t sysfs sysfs /sys*

- exporta informatii din kernel despre echipamente, module kernel, sisteme de fisiere, etc
- subdirectoare

*/sys/block* – linkuri simbolice catre */sys/devices* pt fiecare device din sistem

*/sys/devices* – arborele de structuri de date kernel pt echipamente

*/sys/fs* – contine subdirectoare pt sisteme de fisiere speciale (e.g., *cgroups*)

*/sys/kernel/* - contine fisiere si directoare cu informatii despre starea kernelului

*/sys/kernel/mm* – contine fisiere si directoare cu informatii despre sistemul de gestiune al memoriei

*/sys/module* – contine informatii despre fiecare modul kernel din sistem

samd.

# Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

# Curs 11 - outline

- lucrul in retea
- concepte
  - retea
  - protocol
  - sockets
- exemple de protocoale

# Context

- servicii de comunicare inter-proces (IPC)

- *pipes/FIFOs*

*\$ ls -l | more*

- *sockets*: Unix sau TCP/IP

- diferenta de paradigma: *shared memory* vs *message passing*

- *pipe* – construiește un canal de comunicare în memoria kernel (partajată) a aceleiași mașini

=> consecințe: sincronizare proceselor care scriu/citesc din *pipe* asigurată de kernel, la fel ca și politica de acces la date FIFO

- *socket* TCP/IP – construiește un canal de comunicare între două mașini diferite legate între ele printr-o *rețea de comunicare*

=> consecințe: complexitatea operării rețelei de comunicare (care poate implica mai multor mașini intermediare în efectuarea schimbului de mesaje) impune folosirea unor *protocoale de comunicare*



# Retea

## Intranet sau retea

- alcătuit din noduri (calculatoare) numite *host-uri*
- legături fizice: cabluri de retea, canale wireless
- legături logice: calea prin intermediul legăturilor fizice dintre un nod sursă și unul destinație
- probleme: parcurgere de graf, calea cea mai scurtă, comis-voiajor, rețele de flux (network flows)
- o retea poate fi conectată la una sau mai multe rețele

Internet: set de rețele publice interconectate

# Protocol de comunicatie

Modul de comunicare în retea se desfășoară urmând unul sau mai multe protocoale

- descrie în documente *Request for Comments* (RFC)
- Internet Engineering Task Force (IETF)
- Internet Society (ISOC)
- comitete formate din ingineri și informaticieni
- <https://www.rfc-editor.org/retrieve/>
- **Obs:** protocolul este o specificatie, nu o implementare

# Protocoale de comunicatie

- modelul OSI (Open Systems Interconnection)
  - 7 nivele: *fizic, data link, retea, transport, sesiune, prezentare, aplicatie*
- model alternativ, protocoalele internet (a.k.a. protocoalele Department of Defense, DoD):
  - combina ultimele trei nivele OSI intr-un singur nivel: *aplicatie*
  - combina nivelul fizic si data link intr-un singur nivel: *link*
- conceptual, functioneaza pe principiul *stivei de protocoale*: protocolul de nivel cel mai inalt (*aplicatie*) apeleaza la serviciile protocolului imediat anterior (*prezentare*), samd
  - la nivelul cel mai de jos, datele trimise de protocoalele *data link* sunt trimise peste reseaua fizica
  - la receptie se parcurge calea inversa
- exemple protocoale:
  - data link (retele LAN): Ethernet (CSMA/CD), Token Ring, ATM, Wireless
  - retea: IP
  - transport de date: TCP, UDP, SCTP
  - sesiune: RPC
  - prezentarea datelor: XDR, ASN.1, criptare, compresie
  - aplicatie: DNS, SMTP, HTTP, FTP, SSH, etc
- *socket* : abstractizeaza comunicatia realizata la nivel aplicatie folosind protocoalele de nivel transport ale internetului

# Internet Protocol (IP)

- fiecare host identificat printr-una sau mai multe adrese IP
  - versiunea initială (IPv4) foloseste 32-biti (RFC791)
  - versiunea curentă este pe 128-biti (IPv6)
  - IPv4 în continuare cea mai folosită
  - 32 de biti reprezintă 4 octeti grupati separat prin .
  - interval: 0.0.0.0 – 255.255.255.255
  - anumite subintervale sunt folosite pentru adresare privata (adrese nerutabile)
  - restul adreselor publice sunt accesibile de orice nod din retea
  - **ex:** 192.168.1.1, 10.0.0.1, 216.58.214.238, 8.8.8.8

# Transport Control Protocol (TCP)

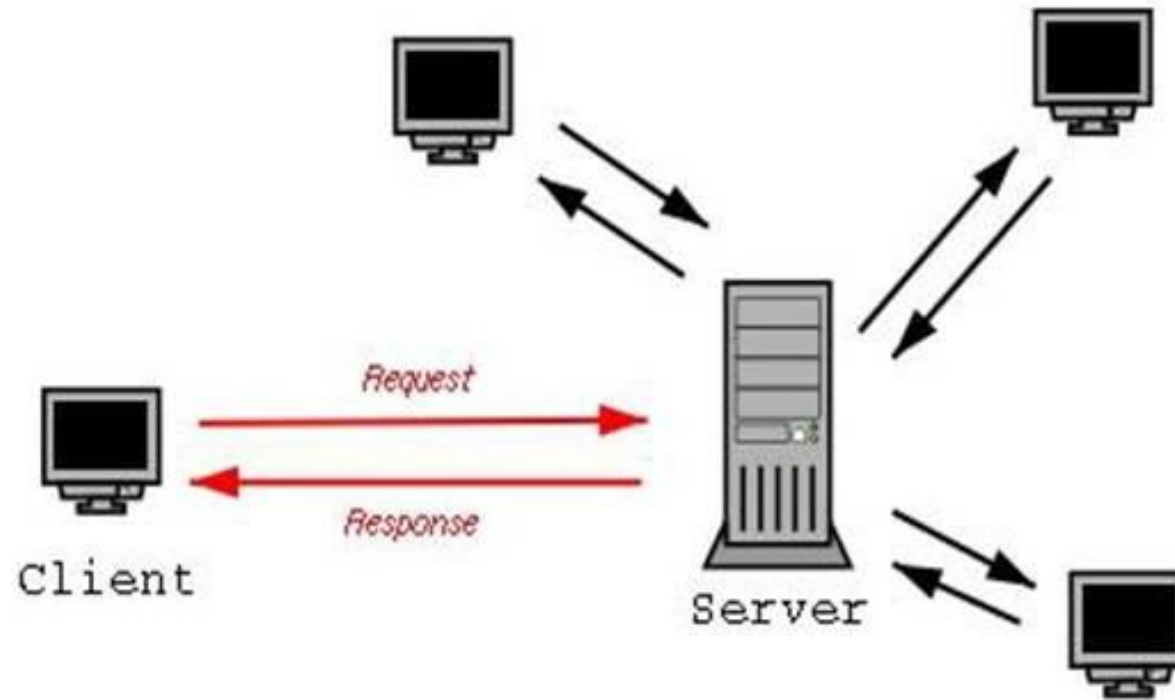
- asambleaza/dezasambleaza mesajele in pachete transmise peste canale de comunicatie eterogene (TCP)
- asigura livrarea pachetelor in ordine la destinatie (TCP)
- rezolva automat probleme de rutare a pachetelor pe cai de comunicatie alternative intre nodurile sursa si destinatie atunci cand anumite masini intermediare nu sunt disponibile (IP)
- asigura retransmisia automata a pachetelor pierdute/defecte (TCP)
- controleaza comunicatia in prezenta congestiei retelei (TCP)
- samd

# Domain Name System (DNS)

Numele asociat unei adrese IP

- mai ușor de ținut minte pentru utilizatorii umani
- protocol descris în RFC1035
- dacă o adresă IP se schimbă, numele rămâne
- inițial toată lumea ținea o agendă locală în `/etc/hosts`
- astăzi procesul e centralizat
- toată lumea folosește aceeași agendă
- mai multe *host*-uri, numite *name servers*, care conțin o copie a agendei
- serverele principale de la care copiază informația restul se numesc *root name servers*
- **ex:** `fmi.unibuc.ro` → `193.226.51.6`
- comenzi în Unix: `nslookup(1)`, `whois(1)`, `dig(1)`

# Arhitectura sistemului distribuit



<http://abcnetworking.wikispaces.com>

# Socket

Comunicarea client-server se face prin *sockets*

- abstractie de nivel sistem de operare
- adresă: adresa IP a unui *host*
- port: intrare a *host*-ului
  - port server: identifica un serviciu (v. */etc/services* pt. well-known ports)
  - port client: identifica procesul client
- adresa si portul formeaza un *endpoint*
- socket: defineste un endpoint
- o conexiune se face intre 2 endpoints folosind protocol comun
- un socket pentru server si unul pentru client
- **ex:** <TCP,192.168.1.6:4444,193.226.51.6:80>



# TCP sockets

- canal de comunicatie (o conexiune) intre doua endpoint-uri (adr IP, port)
- garanteaza livrarea corecta si in ordine a mesajelor intre sursa si destinatie
- mesajele pierdute se retransmit automat
- canalul de comunicatie are o semantica asociata de tip stream de octeti (analog *pipe*)
- ca atare, din socket-ul TCP se citesc octeti, nu mesaje intregi !

# UDP sockets

- canal de comunicatie *fara garantii* intre doua endpoint-uri
- mesajele (numite in acest caz *datagrame*, ca la protocolul IP) se pot pierde, nu se retransmit automat, nu se livreaza in ordine
- mesajele se citesc in intregime, i.e. o datagrama la un moment dat (spre deosebire de TCP)

# Exemple simple protocoale de comunicatie de nivel aplicatie

- *daytime, echo, time*
- *client*: emulat cu comenzile de tip *telnet*

*\$ telnet host port*

- *server*: demoni standard accesibili prin superserverul Internet (*inetd/xinetd*)
  - server cu structura particulara care multiplexeaza mai multe servicii diferite
  - serviciile multiplexate configurate in fisiere de configurare aflate in */etc/xinetd.d*
- in general, serverele au fisiere de configurare, clientii au doar parametri de apel in linia de comanda
  - diferenta e dictata de complexitatea diferita a celor doua tipuri de programe
- serviciile controlate cu comanda *service/systemctl*

*\$ service <serviciu> <start | stop | reload>*

# World Wide Web (www)

- creat la CERN și publicat în 1991
- resurse identificate prin Uniform Resource Locators (URL)
- pagini web prezentate prin hypertext
- HyperText Transfer Protocol (HTTP) RFC2616
- servesc pagini web pe portul 80
- acces criptat prin HTTPS pe port 443 (curând implicit)
- **ex:**
  - `http://fmi.unibuc.ro:80`,
  - <http://fmi.unibuc.ro>
  - `http://fmi.unibuc.ro:443`
  - `https://fmi.unibuc.ro:443`
  - <https://fmi.unibuc.ro>
- browser Unix: `lynx(1)`, `w3m(1)`
- crawler Unix: `curl(1)`, `wget(1)`

# Exemplu comunicatie HTTP

- telnet <host> 80

- apoi comanda

GET / HTTP/1.1

(sau GET /index.html HTTP/1.1)

Host: <host>

<cr/lf>

<cr/lf>

- telnet fmi.unibuc.ro 80

# Email – IMAP, POP

- acces la posta
  - Post Office Protocol (POP) RFC1939
    - POP face o copie locală a mesajelor si le sterge de pe serverul de mail
  - Internet Message Access Protocol (IMAP) RFC3501
    - IMAP operează asupra mesajelor direct pe server
    - IMAP poate tine o copie (partială) local
  - porturi: POP 110, POP3S 995, IMAP 143, IMAPS 993
  - mesageria tinută în format mbox sau maildir
    - mbox: tine toate mesajele într-un singur fisier
    - maildir: tine fiecare mesaj într-un fisier separat

# Email – SMTP

## Expediere mesaje

- Simple Message Transfer Protocol (SMTP) RFC821
- SMTP trimite un mesaj către un destinatar
- protocol vechi, nu cere expeditor → spam, spoofing
- protocol vechi, nu cere autentificare → spam, spoofing
- porturi: SMTP 25, SMTPS
- solutii anti-spam
  - reverse DNS: legătură IP → nume
  - autentificare: utilizator și parolă, criptare
  - Sender Policy Framework (SPF), DKIM (DomainKeys Identified Mail), DMARC (Domain-based Message Authentication, Reporting and Conformance): anti-email spoofing
  - baze de date cu IP-uri de spameri

# Exemplu comunicatie SMTP

```
1 $ telnet mail7.imar.ro 25
2 Trying 193.226.4.17...
3 Connected to mail7.imar.ro.
4 Escape character is '^]'.
5 220 mail7.imar.ro ESMTP Postfix
6 HELO <nume host>
7 250 mail7.imar.ro
8 MAIL FROM: <sender's email address>
9 250 2.1.0 Ok
10 RCPT TO: <receiver's email address>
11 250 2.1.5 Ok
12 DATA
13 354 End data with <CR><LF>.<CR><LF>
14 Subject: test
15
16 Acesta este corpul mailului. Textul trebuie incheiat cu un "." singur pe linie, ca mai jos.
17 Subiectul trebuie sa fie pe prima linie dupa raspunsul serverului la comanda DATA si contine
18 cuvantul cheie "Subject:"
19
20 .
21 250 2.0.0 Ok: queued as ABC6B6A022D
22 QUIT
23 221 2.0.0 Bye
24 Connection closed by foreign host.
```



# File Transfer Protocol (ftp)

## Transfer de fisiere

- protocol vechi din 1971, activ și astăzi RFC959
- autentificare și criptare FTPS
- port separat de comandă (21) și date (20)
- **exemplu:**  
ftp://[user[:password]@]host[:port]/url-path
- comandă Unix: ftp(1)
- **ex:**  
\$ ftp [alex@fmi.unibuc.ro](ftp://alex@fmi.unibuc.ro)  
\$ ftp -a fmi.unibuc.ro  
\$ ftp fmi.unibuc.ro -P 2121

# Peer-to-peer

Tranfer de fisiere distribuit fără server

- servicii bazate pe Distributed Hash Tables (DHT)
- toti clientii transmit si primesc date
- mod eficient de transfer pentru date mari
- elimină presiunea de pe un singur server

# Secure Shell (ssh)

Protocol sigur de conectare la alt calculator

- specificat în RFC4253, port 22
- implementarea ubicuă OpenSSH creată de grupul OpenBSD
- <http://www.openssh.com>
- comandă Unix: ssh(1)  
\$ ssh [user[:password]@]host[:port]
- autentificare parolă, cheie asimetrică etc.
- o dată autentificat se deschide un terminal pe acel *host* în care puteti începe să dati comenzi
- se poate folosi interfata grafică cu argumentul -X
- folosit peste tot în programare si administrare

# Secure Copy (scp)

Copiere prin protocolul SSH

- Secure Copy Protocol (SCP) nu există RFC
- functionează similar cu comanda `cp(1)`
- are în plus prefixat *host-ul*
- comenzi Unix: `scp(1)`

```
$ scp hello.c fmi.unibuc.ro:
```

```
$ scp hello.c alex@fmi.unibuc.ro:code/
```

```
$ scp -r project/ alex@fmi.unibuc.ro:
```

# SSH FTP (sftp)

Functionalitate FTP prin protocolul SSH

- SSH File Transfer Protocol (SFTP) nu există RFC
- functionează similar cu comanda ftp(1)
- profită de autentificare si criptografia de desubt (SSH)
- de obicei tot pe port 22 este servit
- comenzile protocolului SSH diferentiază între tipuri de sesiuni
- comenzi Unix: sftp(1)

```
$ sftp alex@fmi.unibuc.ro
```

# Virtual Network Computing (vnc)

- bazat pe protocolul Remote Framebuffer (RFB) RFC6143
  - protocol simplu, bazat pe transmiterea de primitive grafice de la server la client + evenimente de la client la server
- oferă acces la interfata grafică de la distanță
  - functionează cu Windows, macOS si X (sistemul grafic Unix)
- VNC este cea mai cunoscută aplicație ce implementează și folosește RFB
- clientul se conectează la server pe portul 5900
- comenzi Unix: `vncviewer(1)`, `vncserver(1)`
  - \$ `vncserver :0` – pornește server pe primul display
  - \$ `vncviewer fmi.unibuc.ro`
  - \$ `vncviewer fmi.unibuc.ro:2` – conectează clientul la display-ul 3
  - \$ `vncserver -kill :0` – oprește serverul

# Remote Desktop (rdp)

- Remote Desktop Protocol, proprietar Windows
- functionează similar cu VNC
- oferă în plus acces la sistemul audio si imprimantă
- clientul se conectează la server pe port 3389
- pe Windows clasic e permis un singur utilizator conectat
- pe Windows server se pot conecta mai multi deodată
- comenzi Unix: `rdesktop(1)`, `xfreerdp(1)`
  - \$ `rdesktop 192.168.1.6`
  - \$ `rdesktop fmi.unibuc.ro`
  - \$ `rdesktop -r sound:remote -g 90% 141.85.225.153`

# Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru



# Curs 12 - outline

- compilarea si link-editarea programelor
- depanarea programelor
- gestiunea proiectelor de programare

# De la sursa la executabil

- sursa – fisierul scris in limbaj de programare (ex. C, C++)
- compilator – traduce fisierul sursa in limbaj de nivel jos (limbaj masina)
- obiect – fisier binar rezultat din compilarea unui fisier sursa
- bibliotecă – fisier binar (deja compilat) ce oferă o anumita functionalitate (ex. functii de comunicare in retea)
- linker – leagă obiecte si biblioteci pentru a produce executabilul
- executabil static – toate obiectele si bibliotecile sunt incluse in fisierul executabil rezultat
- executabil dinamic – contine doar datele si instructiunile proprii + functii stub catre apelurile de biblioteca; bibliotecile sunt pastrate in fisiere separate

# Load time

Rezolvarea dependentelor executabilelor dinamice la momentul incarcarii in memorie:

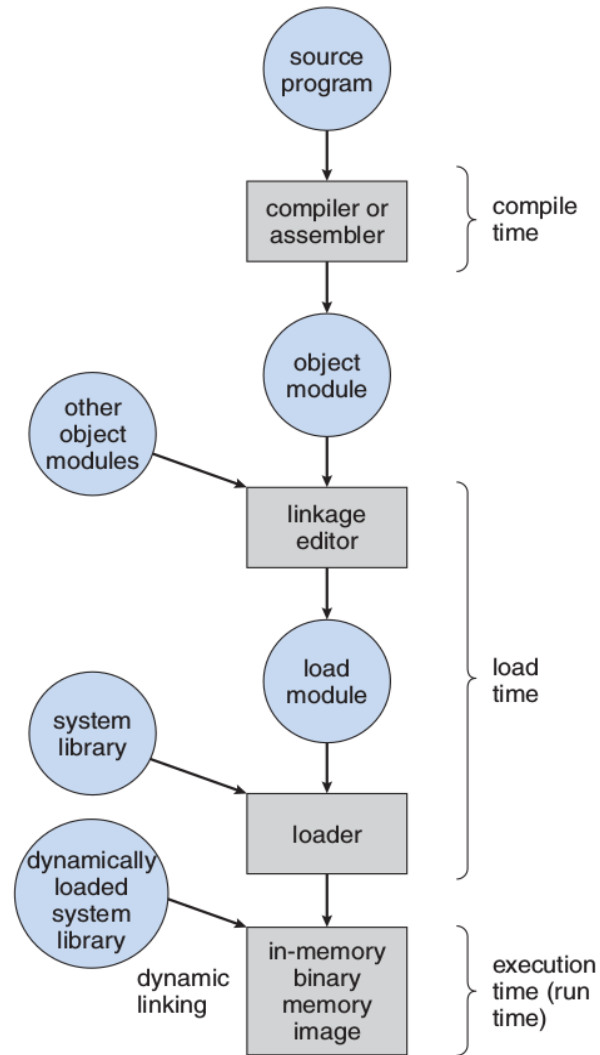
1. sistemul de operare incarca executabilul in memorie
2. inainte de a fi executat se cauta bibliotecile folosite
3. se incarca toate bibliotecile necesare
4. in caz ca nu se gasesc una sau mai multe biblioteci, executabilul este scos din memorie si executia este anulata

# Run time

Rezolvarea dependentelor executabilelor dinamice la nevoie:

1. sistemul de operare incarca executabilul in memorie
2. inainte de a fi executat se caută bibliotecile strict necesare lansarii programului
3. pe parcursul executiei dacă este nevoie de o functie de biblioteca
  1. executia programului este oprita
  2. se cauta in sistem biblioteca necesara
  3. se incarca in memorie
  4. se reporneste de unde a ramas executia programului
4. in orice moment daca o biblioteca nu este gasita si incarcata executabilul este scos din memorie si executia este anulată

# Compile



<http://codex.cs.yale.edu/avi/os-book/>

# Compilatoare

Solutii UNIX:

- gcc – GNU Compiler Collection (C, C++, Java etc)
- clang – LLVM front-end pentru C, C++, Objective C, OpenCL etc.

Indiferent de compilator, in UNIX există comanda `cc` (C compiler) si, optional, comanda `c++` (C++ compiler).

# Executabil simplu

In forma cea mai simpla compilatorul

- primește ca argumente fisierele sursa
- produce executabilul `a.out`
- denumire istorica: formatul de executabil standard pe prima versiune UNIX

```
$ ls hello.c
$ cc hello.c
$ ls
a.out hello.c
$./a.out
Hello, World!
```

Executie prefixata cu `./` - shell-ul cauta executabilul in directorul curent (nu in `$PATH`)

Cu nume de executabil:

```
$ cc -o hello hello.c
$./hello
```

# Obiecte

- generate cu optiunea -c
- compilatorul se opreste dupa producerea obiectului
- nu continua cu crearea unui executabil
- obiectul va fi folosit impreuna cu alte obiecte si biblioteci pentru a produce executabilul final

```
$ ls hello.c
$ cc -c hello.c
$ ls
hello.o hello.c
```



# Biblioteci predefinite

- Bibliotecile externe accesate cu optiunea -l*nume* (**fara spatii!**)  
Obs: numele complet al bibliotecii este *libnume.a* sau *libnume.so*
- compilatorul cauta biblioteca intr-o lista de directoare
- lista este de regula generata si intretinuta de comanda `ldconfig(1)`
- variabila de mediu `$LD_LIBRARY_PATH` contine directoare proprii ce contin biblioteci
- biblioteca gasita este folosita impreuna cu alte obiecte si biblioteci pentru a produce executabilul final

Folosirea bibliotecii de sistem *crypto*:

```
$ cc hello.c -lcrypto
```

# Crearea bibliotecilor statice

- se creeaza o arhiva cu fisierele obiect

```
$ ar -r libhello.a hello.o
```

- eventual se verifica arhiva

```
$ ar -t libhello.a
```

```
$ objdump -p libhello.a
```

```
$ nm libhello.a
```

- se linkediteaza programul

```
$ gcc -o main main.c -L. -lhello
```

- se ruleaza programul

```
$./main
```

# Crearea bibliotecilor dinamice

- se creeaza fisierele obiect in format fpic/fPIC  
\$ gcc -c hello.c -fpic
- se creeaza biblioteca dinamica (shared object)  
\$ gcc -shared -o libhello.so hello.o
- eventual se pune biblioteca in directoarele predefinite ale sistemului (/lib,/usr/lib,/usr/local/lib, samd) si se ruleaza ldconfig
- se linkediteaza programul (cu -L daca nu s-a efectuat pasul anterior)  
\$ gcc -o main main.c -L. -lhello
- se verifica dependentele programului  
\$ ldd main

# Rularea programelor linkeditate dinamic

- daca biblioteca e in rpath (s-a rulat ldconfig, ldd o gaseste)

```
$./main
```

- altfel, trebuie specificata calea catre biblioteca dinamica

```
$ LD_LIBRARY_PATH=. ./main
```

sau

```
$ export LD_LIBRARY_PATH=.
```

```
$./main
```

# Optiuni de compilare utile

Optiuni utile folosite des la compilare

- `-o nume` – numele executabilului sau obiectului rezultat (ex. `hello` in loc de `a.out`)
- `-Wall` – afiseaza toate mesajele de *warning*
- `-g` – pregateste executabilul pentru o sesiune de depanare (debug)
- `-Onumar` – nivelul de optimizare (0 → fara optimizari)
- `-O2` este folosit implicit, `-O0` este folosit adesea impreuna cu `-g` pentru depanare

Ex: compilare `hello.c` cu simboluri de debug, fara optimizare, in executabilul cu numele `hello`

```
$ cc -g -O0 hello.c -o hello
```

# Depanare

Notiuni folosite intr-o sesiune de depanare

- break, breakpoint
  - punct in care sa se opreasca executia
  - poate fi o functie, o linie dintr-un fisier, chiar și o instructiune de asamblare
  - se poate investiga starea variabilelor la momentul respectiv
- backtrace, trace, stack trace
  - apelul de functii care a dus in punctul curent
  - exemplu: `main()` → `decompress()` → `zip decompress()`
- watch, watchpoint
  - punct in care să se opreasca executia *daca este indeplinita o conditie ceruta de utilizator*
  - poate fi modificarea unei zone de memorie sau a unei variabile

Programul cel mai comun folosit pentru depanarea executabilelor este gdb(1).

# Comenzi gdb(1)

## Comenzi uzuale

- `break symbol` – seteaza breakpoint la simbolul respectiv (ex. `main()` sau `hello.c:10`)
- `run` – incepe executia (de regula dupa ce au fost setate breakpoint-urile)
- `next` – o dată ajuns într-un punct de oprire, mergi la urmatoarea instructiune
- `continue` – continua pana la urmatorul punct de oprire sau pana la terminara executiei programului
- `print variabila` – afiseaza valoarea unei variabile
- `quit` – iesire din gdb(1)

# Sesiune de depanare

```
$ cc -g -O0 hello.c -o hello
$ gdb hello
(gdb) break main
Breakpoint 1 at 0x546: file hello.c, line 5.
(gdb) run
Starting program: /home/user/src/hello
Breakpoint 1 at 0x101a49c00546: file hello.c, line 5.

Breakpoint 1, main () at hello.c:5
5 printf("Hello , World!");
(gdb) next
Hello , World!
6 return 0;
(gdb) continue
Continuing.
Program exited normally.
(gdb) quit
```



# Proiecte

Produsele software sunt alcatuite din mai multe componente:

- diferite module (ex. comunicatie, logging)
- diferite biblioteci (ex. compresie, criptografie, http)
- mai multe fisiere sursa (numite si unitati de compilare)
- intre ele apar diferite dependente (ex. modulul de comunicatie depinde de biblioteca http)
- inter-dependentele dicteaza ordinea de compilare
- dependentele si ordinea de compilare descrise formal in fisiere de tip Makefile
- instructiunile dintr-un Makefile sunt executate cu ajutorul comenzii `make(1)`

Obs: Makefile e nume implicit, se poate insa folosi orice alt nume

`$ make -f MyOwnMakefile`

# Makefile

Format fix

```
target : dependency1 dependency2 [...]
 commands
```

- `target` – rezultatul regulii
- `dependency` – ingredientele necesare (deja existente)
- `commands` – comenzile pentru a produce `target`-ul
- **Atentie!** – este un TAB inaintea `commands`
- primul `target` din fisier sau, daca exista, `target`-ul `all` vor fi executate implicit

**Avantaj:** recompileaza doar fisierele modificate!

# Makefile simplu

```
$ cat Makefile
all: hello
hello: hello.c
 cc -o hello hello.c
clean:
 rm hello

$ make
cc -o hello hello.c
$ make clean
rm hello
$ make hello
cc -o hello hello.c
```

# Variabile Makefile

## Variabile utile

- `$@` – numele target-ului curent (partea stanga)
- `^` – toate dependentele (partea dreapta)
- `$<` – numele primei dependente (ex. `dependency1`)
- `.type1.type2:` – target-ul transforma fisiere de tip 1 in fisiere de tip 2 (ex. `.c.o:`)
- `$(VAR:old=new)` – inlocuieste `old` cu `new` in variabila `VAR`

# Makefile complex

```

PROG=hello
SRCS=hello.c
OBJS=$(SRCS:.c=.o)
CC=cc
CFLAGS=-g -O0
INSTALL=install

all: $(PROG)

$(PROG): $(OBJS)
 $(CC) -o $@ $^ $(CFLAGS)

.c.o:
 $(CC) -c -o $@ $< $(CFLAGS)

install: $(PROG)
 $(INSTALL) $< ${HOME}/bin

clean:
 -rm $(PROG) $(OBJS)
```